Sandboxes for Grid Computing

Andersen, Rasmus

Publication date:
2009

Document version
Publisher's PDF, also known as Version of record

Sandboxes for Grid Computing

Ph.D. Dissertation

Rasmus Andersen

Department of Computer Science

University of Copenhagen

Copenhagen, Denmark

Submission date: May 12th, 2009

Acknowledgments

First and foremost I owe a big thanks my adviser, Professor Brian Vinter, who not only arranged my Ph.D. stipend, but also guided me through the project with a never-ending bank of ideas to research problems, always forthcoming to new suggestions, and always finding solutions to administrative problems. Further, Brian has been a great gateway to the world of research; apart from allowing me to tap into his own worldwide network of research partners in academia and corporate research, Brian has also enabled me to establish my own network by letting me attend many international conferences and 3 Ph.D. summer schools with many international researchers from related fields of science.

In the autumn of 2007, Brian introduced me to Prof. John Morrison from the Centre for Unified Computing, University College Cork, Ireland, who was kind enough to host me at his research group during the winter 2007/2008. Despite my rather short stay of only 3 months, John opened many doors for me, and put me in contact with Dr. David Power, with whom I spend many joyful hours discussing our software systems and trying to find common ground for a collaborative research project. I am very grateful to all the folks at UCC, who made my stay very enjoyable.

In our own research group, Jonas Bardino and Henrik Høy Karlsen have been a great help in getting things up and running in our Grid system. My fellow Ph.D. student Mar-

tin Rehr has excelled in delving into my software implementations and suggesting other methods and solutions, and has been a great travel companion at various conferences, work shops, and summer schools.

Last but not least, thanks to my caring, understanding and wonderful wife, Vivi. Thank you for keeping up with my highly asynchronous working hours, all my late Friday discussion sessions, all my work-favored prioritizing, and all my travels, especially the 3 months in Ireland while you were in the final stage of pregnancy.

The work presented here is funded by the NABIIT* project and the Faculty of Science at the University of Copenhagen.

Rasmus Andersen, May 2009

*Programkomiteen for Nanovidenskab og -teknologi, Bioteknologi og IT

Table of Contents

Chapter 1

Introduction

This project focuses on sandboxing techniques for Grid Computing, why sandboxing is essential in a Grid Computing context, and how it can be deployed in an actual Grid system. The main contribution of this thesis is the design and implementation of 2 different sandboxes, each of which make use of a special remote file access library. When put into action, the ultimate goal of these sandboxes is to leverage the potential of Grid Computing by combining it with Public Resource Computing, and making the combined compute power accessible to researchers from other fields than computer science.

The thesis is outlined as follows: This chapter first motivates the use of sandboxes for general scientific exploitation in a Grid system, and then it details the project goals. Chapter 2 presents background material for the thesis. Chapter 3 describes the remote file access library, Chapter 4 and 5 present the designs and implementations of the two sandbox solutions, the MiG Screen Saver Sandbox, and the Scientific Bytecode Virtual Machine, respectively. Chapter 6.1 discusses future directions for the proposed models, and Chapter 6 concludes.

1.1 Motivation

Sandboxing for Grid Computing. Parsing the thesis title from behind, Grid Computing [KF98] is a technology that emerged in the late 1990's as a distributed computing platform for computationally intensive scientific applications, generally known as eScience. Grid Computing arose out of necessity to accommodate the immense requirements of eScience. As personal computers were not enough and supercomputers were much too expensive for small-scale projects, researchers in the mid 90's pooled together personal computers into cluster computers to obtain cheap compute power. The cluster computing paradigm was quickly recognized as the most cost effective approach to scientific simulations. The next step to fit the needs of the ever-expanding eScience requirements, was to pool together several clusters from geographically dispersed sites. Grid Computing then emerged with the promise of making access to compute power from distributed heterogeneous computer resources as easy as the way we have access to electricity in an electric power grid. On the provider side, any type of device with a network interface could in theory be hooked up as a Grid resource, which would then be made accessible for the user side through a Grid middleware.

In the meantime the related Volunteer Computing systems, or Public Resource Computing [Sar98; Wol96; And03], became very popular with many successful academic research projects. In these systems, the compute platform was composed of the public's personal computers; private people with a personal interest the topic of the research project would donate their computing resources to the project.

Today, 10 years later, the situation is more or less the same; Grids are still primarily composed of clusters from various scientific compute centres, and the Volunteer Computing systems are still 'stealing' excess compute cycles from standard desktop com-

puters donated by volunteers. However, the performance of the latter has by far outrun the Grid systems. For instance, at the time of writing, the very popular Folding@Home project operates at whopping 5 PFLOPS, while the RoadRunner supercomputer tops the top 500 supercomputer list with a peak performance of 1.1 PFLOPS. At first sight quite surprising, at second a paradox, since the very idea of the Grid was to share resources on an unprecedented scale. Obviously, and effectively stamped by the success of the Volunteer Computing systems, there is a huge unleashed potential here in getting something as simple as general desktop computers on the Grid.

Thus, the natural extension to the Volunteer Computing scheme is to 'gridify' it. In their current form, primarily based on the BOINC framework [And04], these public resource computing projects are merely one-way systems; one can only donate resources. A resource has a pre-installed client program that continuously fetches new *data* to execute, as opposed to a Grid, where resources continuously fetch new *programs*. The entry cost for a researcher to actually submit new programs to these Volunteer Computing systems is high enough to render them useless for smaller research projects in need of compute power [*].

An important aspect of Grid Computing is to allow seamless access for researchers to the huge amount of connected resources, thus creating the illusion of unified access to one big supercomputer. Gaining widespread acceptance as an important computing platform depends on the ability of making the technology accessible to general exploitation, not only for computer specialists, but also for researchers from other areas, since they are the real end-users of the systems.

'Gridifying' a Volunteer Computing system, i.e. not only making it easy for the

[*]According to the BOINC website, `http://boinc.berkeley.edu/trac/wiki/BoincIntro`, it takes 3 man-months to port an existing application to the BOINC framework and about $5000 for hardware used for project management

public to donate their idle resources, but also enabling researchers to utilize these public resources, gives a whole new dimension to a Grid. However, the two systems differ significantly on two aspects: Security and portability. Firstly, a resource in a Volunteer Computing system only hosts one single application that keeps getting new data, while a resource in a Grid system hosts arbitrary untrusted applications. Hence, there is a substantial difference in the security level to ensure the host system. Secondly, while a 'standard' Grid is typically composed of Linux/Unix desktop computers, clusters, and supercomputers from different trusted compute centres, a volunteer computing grid drastically increases the heterogeneity of the system. For instance, the client statistics from the Folding@Home [LSSP09] project reveal that the biggest contributors are NVIDIA GPUs and PlayStation 3's, markedly trailing ATI GPUs and Windows machines. Linux and Mac machines are barely noticeable. Obviously, porting a single Volunteer Computing application to these architectures is considerably easier than enabling arbitrary programs to run on any architecture.

With the abundance of new powerful architectures in personal computers and devices, we now face many radically different CPU architectures, GPU architectures, operating systems, software environments, and specific limitations to memory, network, and disk usage. Therefore, enabling execution of arbitrary code on connected resources requires a lot of work in specifying an execution environment. Ultimately, before non-computer specialists can adopt this computing platform, it is necessary to hide these complexities of the Grid system, and free the applications from all architectural details of the computational resources connected to the Grid.

To address these issues, the last part of the thesis title comes into action: Sandboxing. A sandbox is a confined environment in which a host system safely can allow untrusted applications to execute; a sandbox can be equipped with a fresh execution en-

vironment, and any malicious behaviour in the sandbox has no way of affecting the host system. However, the increased abstraction level does not come un-billed, as a sandbox incurs a significant performance loss on the applications executing in the sandbox. The thesis will present various ways of mitigating this drawback.

Virtual machines are specific examples of sandboxes, which not only can ensure the safety of host computers, but in effect can help bridge the architectural boundaries in a highly heterogeneous Grid system by raising the abstraction level from the underlying hardware platform. Ideally, the ultimate vision is to be able to move around application code as freely as application data can be moved around.

That said, an important aspect of the virtual machines presented in this thesis, is that they *do not* move around application data, but instead access all application data directly on the Grid storage servers in a remote file access fashion. Traditionally, job submissions in Grids use the intuitive and very simple staging technique, in which all files are transferred to the execution node before job execution, and transferred back to the storage servers upon job completion. However, as elaborated in Chapter 3, many types of scientific applications would benefit greatly from using a remote file access model, thus saving the time used for transferring files. Further, recent trends in network and hard disk technology suggest that it is inefficient to copy data from place to place.

1.2 Project Goal

During the last decade, Volunteer Computing systems have shown the potential of deploying applications from many scientific domains on distributed computing platforms. As these systems sustain petaflops of computing power from millions of connected personal computers, they have outstandingly overtaken what was promised by the Grid

Computing visions. Fully leveraging the computing power of both the Grid Computing and Volunteer Computing paradigms is a huge challenge due to the immense scale and extreme heterogeneity of a combined system.

The goal of the project is to facilitate general scientific applications on a Grid Computing platform composed of typical desktop devices. For the first part, facilitating scientific applications, sandboxing technology in the form of virtual machines, will be essential in order to provide security for the host system when deploying the applications on privately owned computers. For the second part, measures will be taken on how to combine Grid Computing and Volunteer Computing systems.

The thesis will present two types of virtual machines seeking to close the gap between the two computing paradigms. The first solution, named MiG Screen Saver Sandbox, or MiG-SSS, is based on existing virtual machine solutions, while the other, the Scientific Bytecode Virtual Machine, or SciBy VM, is an abstract machine built from scratch to address problems obtained when using existing virtual machine software.

The remote file access module, MiG-RFA will be available as a library for the MiG-SSS, while it will be an integral part of the SciBy VM.

1.3 Results

The results of the main contributions - the MiG-RFA, the MiG-SSS, and the SciBy VM - have been promising. Each of the contributions has been developed, tested, and benchmarked individually with very encouraging results.

The remote file access module, MiG-RFA, has proved the theory that it is inefficient to copy around application data from place to place as opposed to accessing data remotely. In worst case scenarios, the module obtains performance numbers similar

to the staging technique. However, as soon as we turn to scientific applications, the module clearly outperforms the copy semantics, and for special types of applications, speedups of above one thousand is well within the realm of possibilities.

The MiG-SSS has been running reliably in the MiG for a couple of years. It has been downloaded by approximately 1000 willing volunteers who kindly offered their private excess computing power to scientific research. On the user side, several research projects in the fields of molecular chemistry and nano-science have been deployed on the donated resources.

Lastly, while the SciBy VM is still in a premature state, the performance numbers have been very reassuring. Using libraries from a range of scientific application domains, the machine not only outperforms similar virtual machines, but even exhibits native performance in all experiments on a number of different hardware architectures.

1.4 Publications

During the course of this dissertation, the following papers have been published:

- Rasmus Andersen and Brian Vinter, *Performance and Portability of the SciBy Virtual Machine*. Submitted for the special issue on Volunteer Computing and Desktop Grids, Journal of Grid Computing.

- Rasmus Andersen and Brian Vinter, *The Scientific Byte Code Virtual Machine*. The 2008 International Conference on Grid Computing and Applications, Las Vegas, USA.

- Rasmus Andersen, Brian Vinter, David Power and John Morrison, *Supporting MiG & WebCom Interaction*. The Sixth International Conference on Engineering

Computational Technology, Athens, Greece.

- Rasmus Andersen and Brian Vinter, *Direct Application Access to Grid Storage*. Concurrency and Computation: Practice & Experience. Volume 19, Issue 9, June 2007.

- Rasmus Andersen and Brian Vinter, *Harvesting Idle Windows CPU Cycles for Grid Computing*. Proceedings of the 2006 International Conference on Grid Computing & Applications.

The following book chapters have also been published:

- Rasmus Andersen, Martin Rehr, and Brian Vinter, *Cycle-Scavenging in Grid Computing*. Recent Developments in Grid Technology and Applications

- Brian Vinter, Rasmus Andersen, et al., *Towards a Robust and Reliable Grid Middleware*. Recent Developments in Grid Technology and Applications

Chapter 2

Background

eScience, modelling computationally intensive scientific problems using distributed computer networks, has driven the development of Grid technology and as the simulations get more and more accurate, the amount of data and needed compute power increase equivalently. Many research projects have already made the transition to Grid platforms to accommodate the immense requirements for data and computational processing. Using this technology, researchers gain access to many networked computers at the cost of a highly heterogeneous computing platform, as depicted in Figure 2.1. A researcher writes an application, submits it to the Grid middleware, which then finds a suitable resource.

Obviously, maintaining application versions for each resource type is tedious and troublesome, and results in a deploy-port-redeploy cycle. Further, different hardware and software setups on computational resources complicate the application development drastically. One never knows to which resource a job is submitted in a Grid, and while it is possible to assist each job with a detailed list of hardware and software requirements, researchers are better left off with a virtual workspace environment that

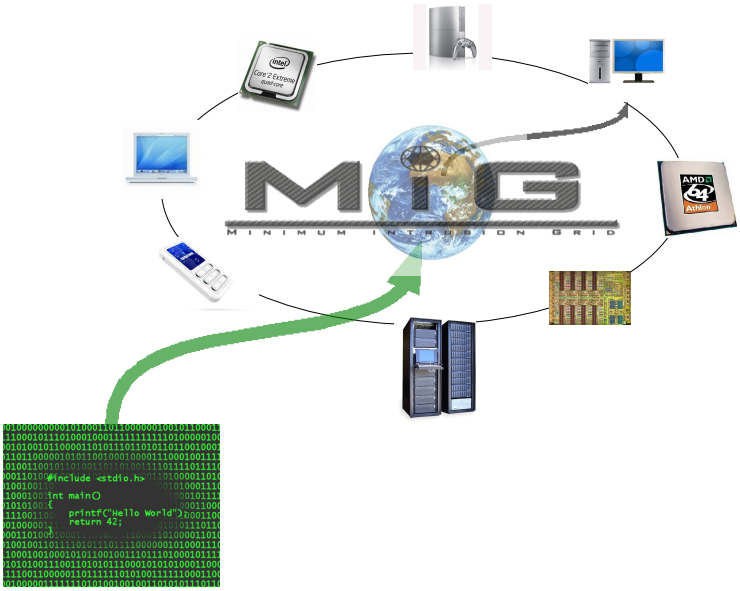


Figure 2.1: Once a researcher submits a job to a Grid composed of many heterogeneous resources, a suitable resource is pointed out to execute the job.

abstracts a real execution environment.

Hence, a virtual execution environment spanning the heterogeneous resource platform is essential in order to fully leverage the Grid potential for researchers with little knowledge in Grid technology. From the view of applications, this would render a resource access uniform and thus the much easier ”compile once run anywhere” strategy; researchers can write their applications, compile them for the virtual machine and have them executed anywhere in the Grid.

The main building blocks in this project arise from properties from virtual machines, eScience, and a Grid environment in a combined effort, as shown in figure 2.2.

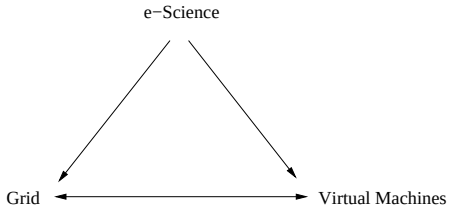


Figure 2.2: Relatationship between VMs, the Grid, and eScience

2.1 Grid Technology

Increasingly complex problems and increasingly powerful high-speed network and processing power technologies are the cornerstones of Grid technology. These factors have incited the Grid from its inception with the initial goal of interconnecting geographically dispersed computer resources over the Internet to create an enormous pool of processing power.

The technology enables the Grid Computing concept that is a form of distributed computing involving dynamic coordination and sharing of a wide variety of heterogeneous resources. Thus, the Grid bonds and unifies remote and distributed resources ranging from supercomputers and special engineering devices to personal digital assistants.

In a wider perspective, Grid technology is going to revolutionize the popular perception of what a computer is, ultimately providing unlimited processing power on tap to everyone.

2.1.1 Reasons for Grids

Despite the ever increasing improvements in network, storage, and processing power technologies, single computational resources fail to keep up with the demands in the fields of engineering, business, and science. To this end, Grid Computing gives the

opportunity to use distributed aggregated resources as one huge unified computing resource.

A number of scientific problems need to be solved by means of simulations requiring an enormous amount of processing power, and specialized scientific devices produce equivalently large data volumes that science communities need shared access to, in order to analyze the data. The former problem domain applies to what is referred to as a *computational Grid*, the latter to a *data Grid*.

A computational Grid is a collection of distributed heterogeneous computing resources that are aggregated to act as a unified processing resource or virtual supercomputer. This aggregation into a unified pool of processing power involves coordinated usage policies, job scheduling and queuing characteristics, Grid-wide security, as well as user authentication and authorization. Computational Grids thus provide efficient processing of CPU-bound applications and allows the collected pool to efficiently service different kinds of jobs.

Data Grids enable users and applications to access and manage data sets from distributed locations securely. Like computational Grids, data Grids rely on security software and usage policies and may span multiple administrative domains. Examples of problems in both types of Grids are given below, in Section 2.1.4.

For institutions, organizations, and business enterprises, the Grid technology provides a means of sharing all kinds of resources and offers a much less expensive alternative to frequently purchasing new costly hardware.

For the end-users, other factors are in play concerning the wide-spread use of the Grid. Nowadays people are forced to buy, update, upgrade and maintain a personal computer. An inherent problem with a personal computer is that most users will never be able to fully use its potential, while its capabilities are insufficient when they are

really needed. Most of the time, these computers are in an idle state wasting processing power that, in a Grid environment, is a valuable commodity.

The Grid provides users unlimited on-demand access to computational services, data services, and application services, i.e. transparent access to remote software. Thus, Grids are moving beyond the academic, supercomputing, realm and into commercial use.

2.1.2 Virtual Organizations

Collaboration is an important issue in today's science. Researchers in science communities, members of a consortium, students at a university are examples of communities that may want to cooperate and interact with colleagues by means of shared access to instrumentation, software, digital libraries, data archives, or computational resources.

For example, in the High Energy Physics project (HEP) at CERN, thousands of researchers distributed all over the world analyze the same data set generated by a single accelerator, the Large Hadron Collider.

Virtual Organizations, abbreviated VOs, are an important point in the Grid. They enable communities to share all kinds of services in a controlled fashion, by letting each resource owner define the local usage policies by means of VOs. For example, all researchers in the HEP project may access the generated data volumes and use certain visualization tools from involved resources, while other communities may use the resources for generic compute cycles only. Thus, members of a VO are authorized to access a set of services on a set of Grid resources.

As shown in Figure 2.3, the Grid provides mechanisms for virtual organizations to include different groups of users and to share resources between VOs.

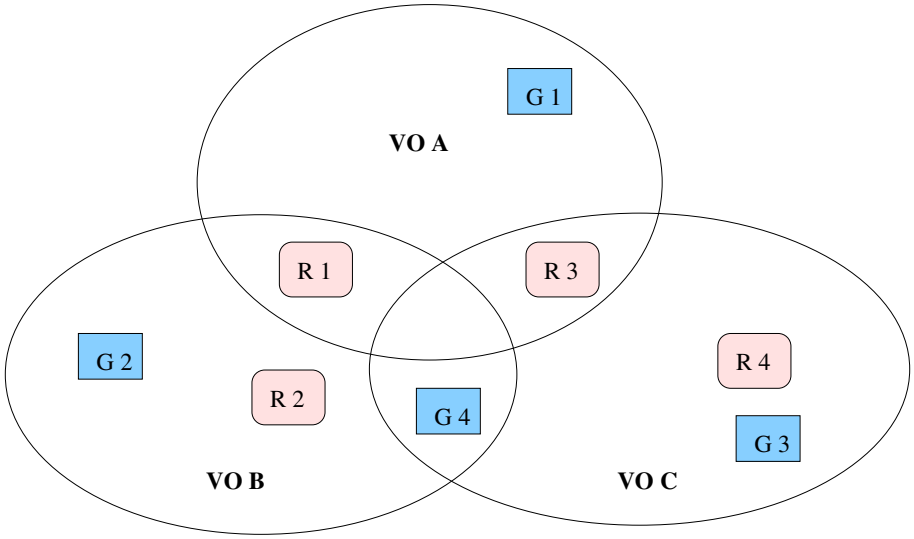


Figure 2.3: Virtual organizations can span multiple groups of people (G), i.e. organizations, institutions, or individual persons, with access to dedicated or shared resources (R).

2.1.3 Grid Architecture

Realizing a Grid first of all requires individual software and hardware components combined into a networked resource. Next we need the deployment of low-level core middleware to provide access to the resources and user-level middleware for the aggregation of geographically distributed resources. Finally, users should be able to submit jobs to the Grid.

These fundamental components of the Grid architecture are organized into layers of different widths, following the principles of the ”hourglass model”, as shown in Figure 2.4. All components in one layer share some common characteristics and are built on the capabilities provided by a lower layer. The width difference indicates that layers at the base and the top may include many components, whereas the narrow neck should only include a minimal set of components.

The base of the hourglass, the fabric, consists of all networked resources, such as

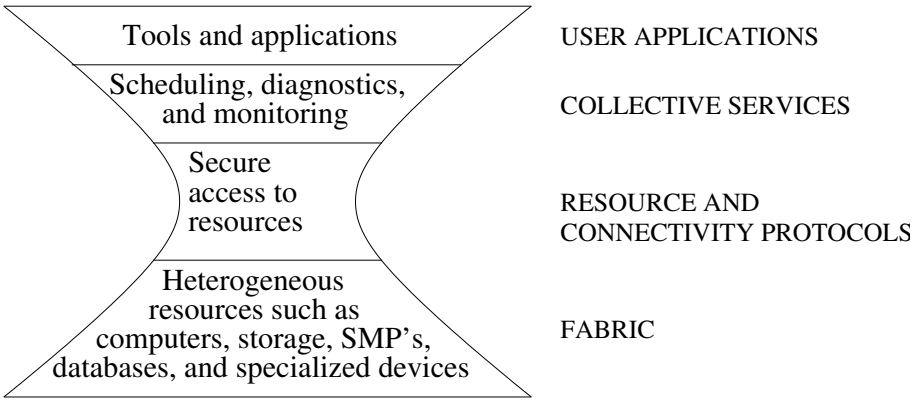


Figure 2.4: The hourglass model

low-end computers, SMPs, clusters, databases, storage devices, scientific instrumentation, etc. These form the collective fundamental services of the Grid.

At a minimum, all resources must implement inquiry mechanisms in order to permit discovery of their capabilities, structure, and current state as well as management mechanisms to allow for control of the service.

The resource and connectivity layers at the neck of the hourglass must be implemented everywhere and, therefore, should only consist of a small number of trusted protocols and services. The connectivity layer defines the basic set of communication protocols for interaction with the underlying fabric layer on the one hand, and authentication protocols for verification of the identity of users and resources, on the other.

The resource layer also interacts with the fabric layer, utilizing its management and inquiry mechanisms to enable secure initiation, monitoring, control, and accounting of sharing operations on a single resource. The security is enforced by the communication and authentication protocols of the connectivity layer.

The aggregated set of protocols in the resource and connectivity layer captures the fundamental mechanisms of sharing across a large pool of different resource devices and provides secure access and control of individual resources. Built upon these, the col-

lective layer contains protocols and services for coordinating collections of resources. A large set of collective services can be implemented, for instance directory services, scheduling, monitoring, diagnostics and data replication services. Also, information about the Grid itself may be encapsulated in a meta computing service in this layer. Thus, the functionality of the collective layer spans from general services to highly domain specific services.

At the top of the system are the user applications. This layer represents the interfacing between users and the Grid. Applications may use Grid-enabled languages and utilities, but the interfacing could also be accomplished by means of Grid portals specific to certain Grid-enabled applications or to users of certain production Grids.

This Grid architecture summarizes the ideal Grid environment: the top layer is provided ubiquitous and seamless access to the base layer such that the underlying differences between networks, platforms, and protocols become completely transparent for the user and the resource. Thus, the intervening layers turn a radically heterogeneous environment into a virtual homogeneous one.

2.1.4 Grid Applications

A wide variety of applications are applicable for a Grid platform. Through experimental study in different testbeds, these application have been categorized into 5 main classes [KF98], listed in table 2.1 and each described in the following sections.

Distributed Supercomputing

Supercomputing is an important and accepted aid in fields of science, business, and engineering. In order to keep up with the ever increasing hardware requirements from

Class	Examples	Characteristics
Distributed supercomputing	Bio-analytical simulations	Applications bound by CPU and memory
High throughput	Folding@Home Cryptographic problems	Harness idle resources to increase aggregate throughput
On demand	Medical instrumentation Cloud detection	Remote resources integrated with local computation
Data intensive	Sky survey Physics data Data assimilation	Synthesis of new information from many data sources
Collaborative	Collaborative design Data exploration Education	Support work between multiple participants

Table 2.1: The 5 main classes of Grid applications

these fields, distributed supercomputing, i.e. deploying aggregated physically distributed supercomputers, emerged as the initial stage of the Grid.

The challenge in distributed supercomputing is porting existing supercomputing algorithms to a Grid environment, where the algorithms may scale to tens or hundreds of thousands, or even millions of nodes. Furthermore, as the difference in latencies between distributed supercomputers might be several orders of a magnitude, the algorithms should be more latency-tolerant and maintain a high level of performance across heterogeneous resources.

High-Throughput

Harnessing idle resources and putting them to work is one of the fundamental ideas behind the Grid. In high-throughout computing, a huge number of embarrassingly parallel tasks are scheduled with the goal of using otherwise idle processor cycles. These tasks are generally huge in number but typically simple in individual size and calculation, but

combined to one monolithic application to be executed at a single site, it would take life-times to finish.

Many such projects exist, for instance the "Screen Saver Science" systems SETI@-Home and Folding@Home. Folding@Home is a distributed computing project which studies protein folding and misfolding. Understanding protein folding completely, could wipe out diseases such as Alzheimer's disease, cystic fibrosis, and many types of cancer.

Folding@Home is one of the biggest distributed computing projects in the world with the biggest distributed computing cluster. In 2007, the project sustained a performance level higher than one petaflops, thus becoming the first computing system of any kind in the world to break the petaflops barrier.

At the time of writing, approximately 250.000 CPUs, mainly owned by private persons distributed all over the world, contribute to the protein folding analysis by letting client software utilize idle CPU-cycles for the analysis.

Note that while this is an example of a Grid application, the Folding@Home project is deployed in a Volunteer Computing system, not a Grid Computing system.

On-Demand Computing

On-demand applications are characterized by an immediate need for a resource that, for a number of reasons, is not available locally. These resources are typically specialized hardware that by concern of high cost of ownership cannot be placed locally.

Generally, supply and demand for such specialized resources differ significantly from standard resources, making Grid-related issues like resource location, scheduling, fault tolerance, and payment for this kind of application extremely important.

As an example of an on-demand application, the Aerospace Corporation developed a system that processes data from meteorological satellites and dynamically acquires

supercomputing resources to deliver the results from a cloud detection algorithm to meteorologists all over the world.

Data-Intensive Computing

Applications in the class of data-intensive computing are characterized by using large distributed data repositories for the synthesis of new information. These applications are mainly I/O-bound, but some may exhibit balanced or dynamic use of computation and communication.

From a Grid perspective, the focus of this kind application is on the massive amount of data flowing through different segments and hierarchies. Data transfers must adapt dynamically to varying latencies and fault situations. Also, optimal use of striping and parallel file systems is an important issue in this area.

The HEP experiments at CERN, where many terabytes of data are generated daily and thousands of scientific collaborators distributed all over the world access these data, is an example of a data-intensive computing application.

Collaborative Computing

Collaborative applications are primarily concerned with enabling collaborative interaction in shared simulations, data repositories, or other computational resources.

The Grid architecture must be able to cope with this kind of real-time requirement, where a wide variety of collaborative interactions can take place.

The possibility of collaborative interactions between colleagues is one of the major advantages of the Grid, known as knowledge services.

2.1.5 Grid Middleware

The Grid architecture maps out a Grid infrastructure where users and applications are transparently linked to computer resources, i.e. users are unaware of the physical location of the resources. According to this architecture, Grid middleware is basically everything between the application and the fabric layer in the hourglass model in Figure 2.4.

From the fabric layer perspective, Grid middleware is the glue that connects all resources, and from the user perspective, it hides all the underlying complexities of the Grid and turns a radically heterogeneous environment into a virtual homogeneous one.

Analogously to the Internet, where the Internet Protocol provides the low-level services in the Internet, Grid middleware is the "Grid Protocol" that provides all basic services required to construct a Grid.

2.1.6 Grid Middleware Services

The middleware consists of a set of Grid services that constitute the design features of a Grid implementation. Since some of these services are located in the collective layer of the hourglass, there may be large number of services.

The most fundamental services that are required in all Grid solutions include job execution services, resource management and scheduling services, information services, security services, and file transfer services:

Job Execution Services First of all, mechanisms to asynchronously submit a job is required. This service includes unique job identifiers as well as authentication and authorization of users.

Resource Management and Scheduling Services Since users are not necessarily aware of which resources are executing their jobs, a mechanism to manage all resources and schedule applications that need to use the resources most effectively is essential.

Information Services This component is responsible for obtaining information about all resources and services on the Grid. Many types of information are viable due to the dynamic nature of Grid.

Security Services Security is of huge importance in the Grid. Clearly, no resource owners are willing to join the Grid, if it introduces security holes to their systems or impacts the usability. The security infrastructure protects all resources and data by means of authentication, authorization, and encryption.

File Transfer Services In order to fully utilize the transparent use of resources, a mechanism is needed to transfer files securely from one location to another.

Based on these fundamental services a wide variety of services can be introduced at a higher level. For instance, as proposed, an advanced file transfer service that adapts dynamically to varying network latencies and minimizes the network load by caching data.

2.1.7 Existing Middleware

The Globus Toolkit

The Globus Toolkit [Fos05] is currently the main core middleware implementation for Grid computing environments. The Globus architecture, shown in 2.5, includes 3 au-

tonomous components that implement the fundamental services of a Grid environment: resource management, data management, and information services. All components are accessible through the security layer. These components provide the basic capabilities and services required to construct a computational Grid.

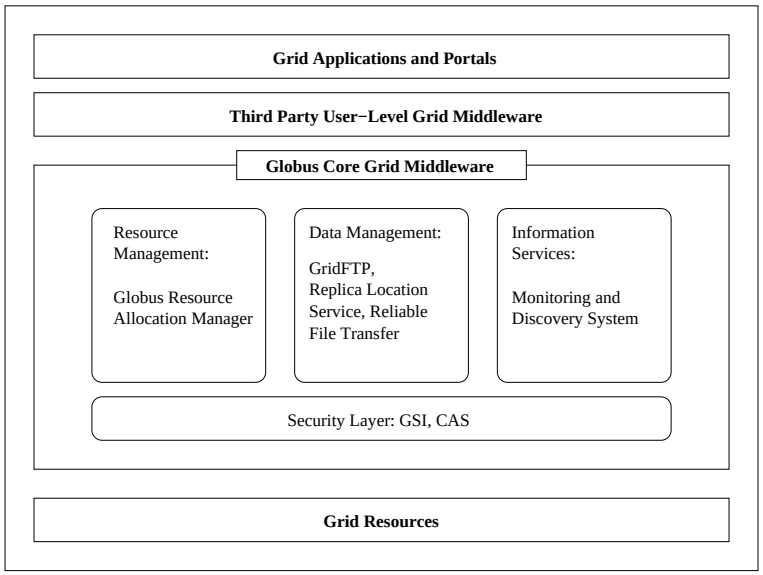


Figure 2.5: The Globus Toolkit Architecture

Security Layer The security layer consists of a Grid Security Infrastructure (GSI) component and a Community Authorization Service (CAS) component.

The Grid Security Infrastructure uses public key encryption, X.509 certificates, and the Secure Sockets Layer (SSL) communication protocol to provide secure authentication and communication. The certificate is a central concept in the infrastructure. All users and services are uniquely identified by a private certificate that is vital to identification and authentication of the user or service. Also, the certificates are used to provide single sign-on for users and services as well as mutual authentication.

The Community Authorization Service (CAS) allows resource owners to specify

local control policies in terms of virtual organizations. All access policies for a given VO are collected at a corresponding CAS-server that users of the VO must contact to gain the right to perform a request.

Resource Management The Grid Resource Allocation and Management provides remote submission and control capabilities. GRAM interfaces to local job scheduling systems by means of a single common protocol and API for requesting status and using remote resources.

As such, the heterogeneity of the local systems, i.e. the variety of different local queuing system, reservation systems, schedulers, and control systems, are encapsulated in the GRAM protocol, thus sparing users from being bothered with the varying resource environments.

However, GRAM introduces some fundamental, severe problems. Firstly, the interfacing to local resource management mechanisms is implemented by a gatekeeper and a job manager. User jobs are submitted to the gatekeeper that authenticates the user and starts a job manager corresponding to the local scheduler. The job manager then submits the user job to the local scheduler. Hence, several levels of scheduling are in play, and since the Grid scheduling depends on the local scheduling, it is merely a job placement, in which neither fairness for users nor optimal utilization of the resources connected to the Grid is provided.

Secondly, the job manager uses the monitoring facilities of the local scheduling system to report the resource status. Upon each user job submission, the Grid scheduler needs to contact all job managers in order to submit the job to the resource with the lowest time to execute. Clearly, inquiring all resources, each potentially including several job managers, does not scale to the size of a Grid.

Finally, this model has a built-in race condition, since two jobs submitted simultaneously will be sent to the same resource with the shortest submission queue, but only one will get the expected queue slot.

Data Management The data management tools are concerned with data movement and data replication. Data movement is achieved using, on the one hand, the GridFTP protocol for secure, robust, fast, and efficient transfer of data, and on the other, the Reliable File Transfer Service that provides a web service interface and supports resuming transfers from clients that disconnect by keeping the transfer state in reliable storage.

Data replication is achieved by the Replica Location Service. Upon creation of files, users register the files in the RLS registry. The RLS then maintains a mapping between the logical file name and one or more physical file names. Hence, the RLS server is queried to find the physical location of a logical file name, and vice verse. In order to achieve scalability and avoid a single point of failure, the RLS server is distributed among the replica catalog servers.

Information Service Obtaining, indexing, archiving, and distributing information about a Grid is handled by a set of information service components, collectively referred to as the Monitoring and Discovery Service, MDS.

The MDS is a three-tier system, consisting of Information Providers, a Grid Resource Information Service (GRIS), and a Grid Index Information Service (GIIS). Information providers exist in all resources and utilize local resource status mechanisms to obtain resource properties and status, such as current load status, operating system, hardware configuration, etc. This information is reported to the GRIS daemon that runs on every single resource. The top-level GIIS is a bunch of Grid-servers forming a hier-

archical distributed system itself. Every GIIS server indexes and archives information from all GRISs and other GIISs connected to it.

The system is based on OpenLDAP and follows a pull model: The information providers generate LDAP entries that, upon a client request, are pulled from a GIIS through the GRIS, optionally using caches. Hence, the snapshots of the current total load provided by the information system that are used in monitors and job submission brokering, are as old as the time it takes to generate the LDAP entries on all resources, parse them at the GRISs, pull them to the respective GIISs and assemble a total snapshot from all GIISs.

Condor

Condor [LLM88] is a full-featured system aimed at utilizing all resources available to the network whenever they are available. By means of job *checkpointing* to enable job migration, and *remote procedure calls* (RPC) to enable a *mobile sandbox* environment where all I/O calls are redirected to the server holding the files, Condor not only manages dedicated compute resources, but also effectively scavenges and manages wasted CPU cycles from otherwise idle resources across the network.

Condor includes its own resource management, job management, resource monitoring, and scheduling policies, and forms a Grid middleware in conjunction with the Globus Toolkit that provides the basic mechanisms for secure communication and standardized access to a wide variety of remote batch systems.

As shown in Figure 2.6, which serves as a general diagram for many emerging Grids, the Condor software consists of two independent components: Condor and Condor-G, the former being the full-featured resource management system, the latter being responsible for reliable job submission and job management.

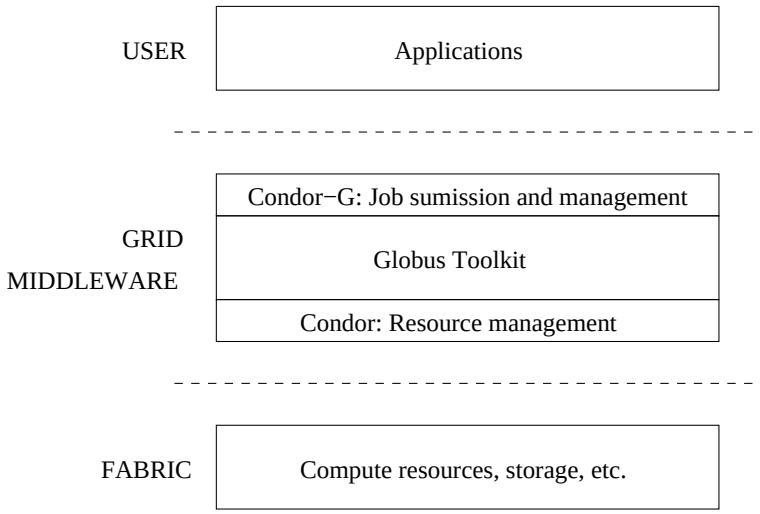


Figure 2.6: Condor in the Grid architecture

Job Submission and Scheduling Job submissions in Condor-G take place by means of an *agent* and a *matchmaker*. Figure 2.7 shows the data flow in Condor. A user submits a job to an agent, who is responsible for storing the job while finding an appropriate resource to run the job (1). Finding an appropriate resource, requires that the agent advertises the requirements and characteristics of the job in so-called *classified advertisements* (ClassAds) to a matchmaker. Resources also advertise themselves using ClassAds to the matchmaker (2). ClassAds are basically a set of uniquely named attributes that the matchmaker scans to find suitable pairs of matching jobs and resources. Finally, the agent and the resource are notified about their match (3) and can now cooperate on executing the job (4).

Analogously to the Globus Toolkit middleware, scheduling in Condor is merely a job-placement mechanism; once submitted to a resource, the job is submitted to the local site scheduler, where the timetable is unknown. Publishing the timetable as an attribute in the resource ClassAd helps planning, but scheduling is still based on a snapshot of the past.

Another approach addresses the job-placement problem more effectively. In *scheduling within a plan*, an agent claims a resource in advance and can then create a schedule for running its own jobs.

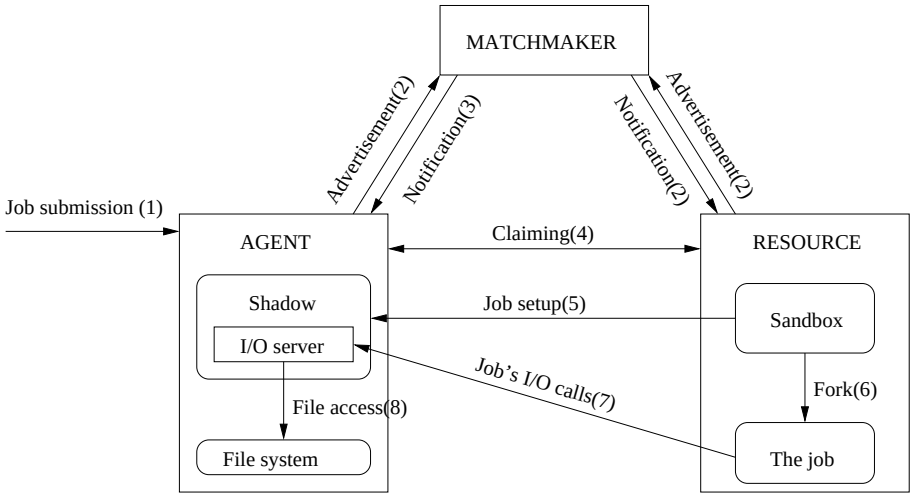


Figure 2.7: Data flow in Condor

Execution environment The execution environment is defined by a *shadow* and a *sandbox* that altogether constitute a *universe*. At the agent storing the job, a shadow is responsible for providing all the details of the job: the input files, the executable, the arguments, etc (5). At the resource, the sandbox provides an execution environment allowing the job to run and protects the resource from malicious jobs (6).

The standard universe emulates all standard system calls in order to provide remote file access through a secure RPC channel to the user's home catalog managed by the shadow (7,8). This emulation requires all user jobs to be relinked with the Condor library.

Check-pointing is a very useful feature of the universe. It provides job migration using the ability to take a snapshot of a running job and store it in stable storage. In

the event of failure or job preemption if a user returns to his workstation, the checkpointing snapshot allows the job to migrate to another idle resource, where it can use the snapshot to reconstruct the process and resume it right from where it left off.

WebCom

Problem solving for parallel systems traditionally lay in the realm of message passing systems such as PVM [GBD+94] and MPI [GFB+04] on networks of distributed machines, or in the use of specialised variants of programming languages like Fortran and C on distributed shared memory supercomputers. The WebCom System [MPK03] relates more closely to message passing systems, although it is much more powerful. Message passing architectures normally involve the deployment of a codebase on client machines, and employ a master or server to transmit or *push* messages to these clients.

Technologies such as PVM, MPI and other metacomputing systems place the onus on the developer to implement complete parallel solutions. Such solutions require a vast knowledge on the programmer's part in understanding the problem to be solved, decomposing it into its parallel and sequential constituents, choosing and becoming proficient in a suitable implementation platform, and finally implementing necessary fault tolerance and load balancing/scheduling strategies to successfully complete the parallel application. Even relatively trivial problems tend to give rise to monolithic solutions requiring the process to be repeated for each problem to be solved.

WebCom removes much of these traditional considerations from the application developer; allowing solutions to be developed independently of the physical constraints of the underlying hardware. It achieves this by employing a two level architecture: the computing platform and the development environment. The computing platform is implemented as an Abstract Machine (AM), capable of executing applications expressed

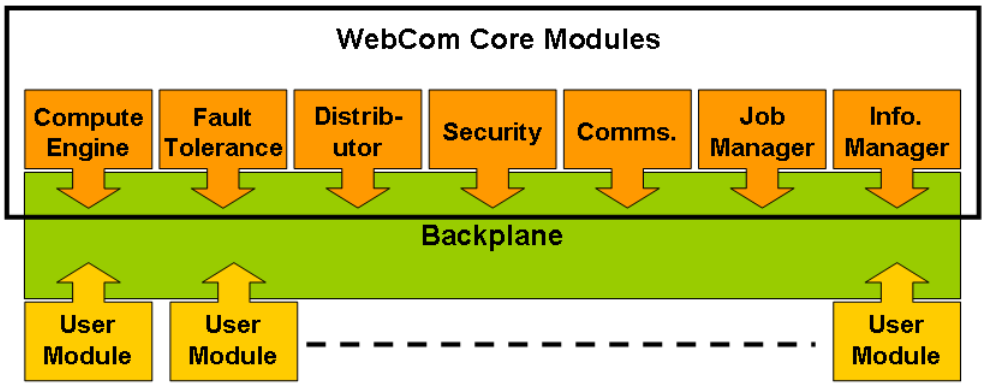


Figure 2.8: WebCom Abstract Machine Architecture showing how WebCom modules plug into the Backplane module.

as Condensed Graphs. Expressing applications as Condensed Graphs greatly simplifies the design and construction of solutions to parallel problems. The Abstract Machine executes tasks on behalf of the server and returns results over dedicated sockets. The computing platform is responsible for managing the network connections, uncovering and scheduling tasks, maintaining a balanced load across the system and handling faults gracefully. Applications developed with the development environment are executed by the abstract machine. The development environment used is specific for Condensed Graphs. Instructions are typically composed of both sequential programs (also called atomic instructions) and Condensed nodes encapsulating graphs of interacting sequential programs. In effect, a Condensed Graph on WebCom represents a hierarchical job control and specification language. The same Condensed Graphs programs execute without change on a range of implementation platforms from silicon based Field Programmable Gate Arrays[MOH03] to the WebCom metacomputer and the Grid.

Architecture Overview The WebCom abstract machine is constructed from a set of modules that plug into a module called the *backplane*. Each module in the WebCom

system falls into one of two categories: it is either a *Core* module or a *User* module. Modules are loaded based on a initial configuration, thus bootstrapping the computational platform. Core modules include the *Compute Engine*, *Fault Tolerance*, Scheduling and Load balancing via the *Distributor*, *Security*, *Communications*, *Job Management* and *Information Management*. User modules can be provided to add additional functionality to the WebCom abstract machine. This architecture is outlined in Figure 2.8.

Compute Engine: The compute engine is responsible for low level task execution. Task composition varies depending on the compute engine in use. The default Condensed Graphs compute engine is responsible for executing applications expressed as Condensed Graphs.

Fault Tolerance: The fault tolerance module detects and corrects faults that occur within the Abstract Machine. Mechanisms in place for fault tolerance range from simply rescheduling failed tasks to employing a unique processor replacement strategy.

Distributor: The distributor is responsible for allocating work to clients. The distributor operates according to a set of installed policies. Typically, a system-wide default policy exists that allows the distributor to select clients for task execution. These policies are plugable and hierarchal in nature and specify items such as the selection algorithm and associated configuration parameters. Nodes within a Condensed Graphs application are executed according to the installed policy. In addition, nodes themselves can specify their own distribution policy. Such nodes will be allocated to clients based on that policy. This provides great flexibility within the abstract machine, as an application is not tied to one particular scheduling algorithm. Different nodes in a single

application can be scheduled using different algorithms. It is possible that multiple applications executing on a single abstract machine can execute using multiple selection algorithms within the distributor.

Security: The security manager is responsible for authenticating the tasks, results and other messages transmitted throughout the WebCom infrastructure. The current security manager is based on the *Keynote*[Ker99] standard.

Communications Manager Module: The communications manager is responsible for communicating tasks to clients. Once the distributor has decided where to allocate a task, it is placed in a queue for the selected client. The communications manager module serves this queue, transmitting the task to the client. Tasks are sent based upon a *Pull Request* mechanism. When a client is willing to accept work, it issues a pull request. A WebCom will respond to this request by *pushing* the task to the client.

Job Manager: Each application within the WebCom abstract machine executes within its own job space. The job manager [CHPM07] can be used to monitor the progress of job execution, pause and restart jobs and also suspend jobs.

2.2 Minimum intrusion Grid

Several Grid middleware systems have been developed and many are currently evolving. Most of them are descendants of, or are based on, the Globus Toolkit [Fos05; EEH+03; dANV+05], and thus suffer from several drawbacks [Vin05; Vin07].

The Minimum intrusion Grid, MiG, is a stand-alone approach to Grid that does not depend on any existing systems. The philosophy behind the MiG is to provide a

Grid infrastructure that imposes as few requirements on users and resources as possible. Users and resources should only install a minimum amount of software in order to access and join the Grid. MiG has been running since 2005 and many applications from various research projects have been deployed [MP05; WVB05; VABK06] on resources ranging from Java applets, via PlayStation 3s and PCs, to cluster computers [RV07; RV08a; AV06; RV08b].

2.2.1 The Four Rules of MiG

The following four rules of MiG constitute the cornerstones of the philosophy behind the MiG system and are in constant contemplation during the design phase:

1. *Minimum intrusion rule:* Nothing produced by MiG can be required to be installed on resources or clients

2. *Programming language:* Everything in MiG must be implemented in one single programming language unless another language is absolutely required. Python is chosen [KV05].

3. *User convenience:* Any design and implementation decision must optimize towards transparency for the users

4. *Software development rule:* Anything that is not right must be thrown away

2.2.2 MiG Design

The main challenge in MiG, is to let the desire for minimum intrusion on users and resources coexist with all the properties, features, and services of a commodity Grid. The following design criteria should be included in the MiG middleware:

Non-intrusive The requirements and the amount of software to be installed should be minimal to users and resources.

Scalable As the Grid could comprise millions of resources and users, scalability is inherently important in any Grid middleware.

Autonomous Grid software should be autonomous, i.e. users and resources should not be affected by middleware software updates.

Anonymous Users and resources should be anonymous to each other, i.e. all interfacing is done through the Grid and they do not see the identity of each other.

Fault Tolerance In a Grid system comprising a large number of users and resources, failure is unavoidable and must be handled gracefully. Failing jobs, resources, Grid servers, Grid processes, and network connections must be transparent to the users and should not affect the operability of the Grid.

Firewall Compliant Machines behind a firewall are not required to have new ports opened in the firewall.

Security Security is fundamental to any successful Grid implementation. Several mechanisms must be introduced to achieve this goal, and all Grid services must integrate a security mechanism.

Secrecy Secrecy is partly achieved by the anonymity property that ensures that resources running a job on behalf of a user, are unaware of the identity of the user. Fur-

ther, means of keeping user files secret to the owner of the resource running the job are necessary.

2.2.3 Architecture

The architecture of the MiG model is a classic client-server approach, where users and resources initiate all communication: the user sends jobs to the Grid, and the resource sends job requests to the Grid and receives a job for execution. Afterwards, the resource sends the result to the Grid, the user is notified and may finally retrieve the result.

To achieve the design criteria of non-intrusiveness, the communication protocols should be trusted and widely used. Furthermore, since the amount of software to be installed should be minimal and no reconfiguration of firewalls should be required, the obvious solution is to take advantage of commonly installed software. Thus, for the interface between users and the Grid, we choose the HTTPS protocol, and for the interface between resources and the Grid, we choose HTTPS and SSH.

The only requirement for users to join the Grid, is that they acquire a certificate for proper authentication when contacting the Grid. Jobs are then submitted through a web browser importing the certificate.

At the other end, resources also need a certificate and, in addition, they need to create a Grid account. User jobs are then run incognito on remote resources as this generic Grid-user.

Thus, the requirements on users and resources are kept at a minimum, as shown in table 2.2

Furthermore, since we have to avoid reconfiguration of firewalls in front of resources, and we do not require any intervention from the local system administrator

Requirement	User	Resource
Certificate	Yes	Yes
Outbound HTTPS	Yes	Yes
Inbound SSH	No	Yes

Table 2.2: User and resource requirements to access and join the Grid.

to install Grid software, we use the SSH protocol and the Grid account to install and maintain the Grid software on the resource. This software is necessary for requesting, initiating and returning user jobs.

Anonymity is inherent in this model, shown in Figure 2.9, since users and resources only communicate with the Grid, while autonomy is achieved by placing all functionality on the Grid servers. By keeping the functionality on Grid servers between the users and resources, the Grid system acts to its clients as a centralized black box that can be maintained and updated without interventing or affecting the clients. In effect, having a fat Grid and thin clients lowers the requirements on users and resources.

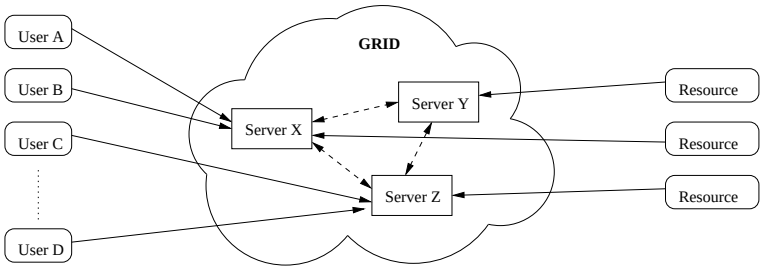


Figure 2.9: The MiG model

As all functionality is centralized on the Grid servers, all workload must be distributed among the servers to ensure scalability. A level of redundancy in this internal distributed system is needed to cope with internal failures. For instance, job submissions are only acknowledged when the job has been successfully placed at a predetermined number of servers, i.e., for a replication rate of 3 in Figure 2.9, one user is assigned

server x,y,z as primary, secondary, and tertiary “job-server”, respectively, while another is assigned server z,y,x respectively.

Keeping the amount of software and protocols at a minimum eases security issues significantly. The chosen communication interfaces, SSH and HTTPS are well-documented and trusted protocols, which we can readily rely on.

Components

On top of the fundamental architecture, the functionality of MiG is built by multiple autonomous components.

Storage

The MiG model introduces home-catalogs for all Grid users, and all file references are relative to this home-catalog. This eliminates all issues with replica catalogs and users are freed of the annoying burden of explicitly uploading input files and downloading output files from specific Grid storage elements. All personal files are referenced directly and accessed through one simple access entry.

Storage is integrated in the Grid system, and the component includes a simple distributed replica backup mechanism to ensure scalability and fault tolerance.

Scheduler

Optimal utilization of available resources and fairness for users submitting jobs are two fundamental issues that are handled in the scheduler component. MiG handles scheduling quite differently than other middlewares, in that the local scheduling precedes the Grid scheduling. Thus, a job is only submitted to a resource once the resource has an

available slot for executing it. In contrast, many other middlewares submit a job to resource on which it is subject to another level of scheduling.

An individual resource schedules Grid-jobs locally. These local Grid-jobs are actually containers for a job, i.e. when a Grid-job is ready for execution, the job itself contacts the Grid scheduler to request a job for the vacant container. The job is then guaranteed to be executed immediately since the Grid-job is currently running. If no jobs are available or applicable, the resource receives an empty job, that instructs the resource to sleep for a while and try again.

This design is illustrated in Figure 2.10. Initially, a user submits a job (1). In the meantime, resources notify the scheduler of vacant slots (2). The scheduler decides which resource fits the job best (3), and the resource receives that job for the vacant slot (4). When the job is done (5), the output files are sent to the storage server on which the user's home catalog resides (6). Finally the user is informed (7) and may retrieve the results (8).

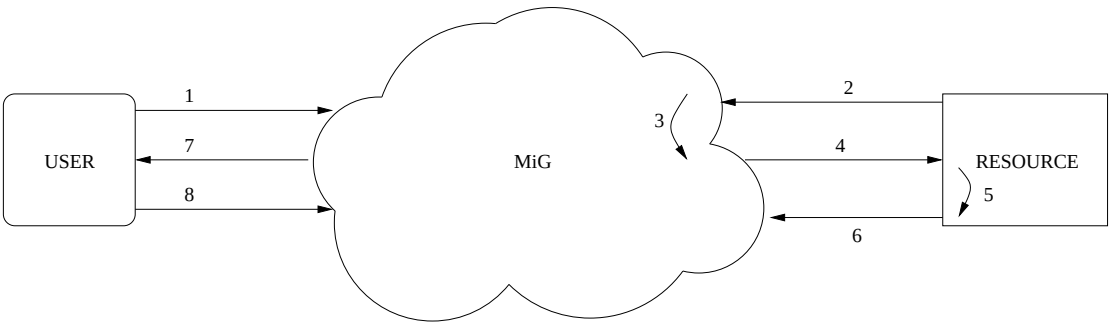


Figure 2.10: MiG Design

Due to the anonymity between users and resources, the job submissions contain all resource constraints, such as architecture, run-time environment, pricing, and memory requirements. All these factors must be taken into account by the scheduler.

2.3 Sandboxing

A sandbox is a security mechanism to separate the execution of an untrusted program in a confined environment, subject to a set of well-defined policies. Once running in the sandbox, it is impossible for the program to damage anything outside the sandbox. The ability to provide a private, isolated, and reliable environment makes sandboxes ideal containers for Grid jobs executed on personal desktop computers.

Today, the term sandbox has a much broader meaning, than when it was originally introduced in software-based fault isolation as `address sandboxing` [WLAG93], in which all unsafe instructions, typically *store* and *jump* instructions, were surrounded by code that would mask the upper bits of the target addresses to prevent access to illegal segments. As sandboxing introduces some code to supervise or control a running program, the execution time is increased. For instance, as a baseline for sandbox overhead, the very basic address sandboxing mechanism only required two arithmetic instructions prior to each unsafe instruction, and the use of five dedicated registers, yet the performance overhead on the `C SPEC92` benchmark suite was 4%.

The literature contains many sandbox variants for different architectures. Untrusted applets [LY99] running in Web browsers, memory leak testing in Valgrind [NS03; NS07], and the Linux `chroot` utility are some typical sandbox examples. The vx32 [FC08] would be of primary interest for a Volunteer System since it has minimal intrusion on the host system, thus lowering the workload for the people trying to donate their resources. Where many similar systems require modifications to the host system, for instance kernel modifications, special privileges and permissions, vx32 is OS-independent and runs native x86 code on unmodified host systems. Applications just need to be linked to a user-level library, which then sandboxes the application in a secure execution environ-

ment. Security is obtained by intercepting all system calls, and by preventing access to memory outside the sandbox. Vx32 is limited to x86 architectures.

An upcoming interesting sandbox is Google's Native Client [YS08]. Aimed at browser-based applications, the system tries to run compute-intensive applications in the address space of the browser at native speed. When an application is sent to a browser, a Native Client plugin loads the Native Client container, which is a sandbox containing native libraries enabling the application to execute at native speed. Like vx32, Native Client is limited to x86 architectures, and since the workload required to express all types of security restrictions in native machine code is quite substantial, porting to other architectures is unlikely to happen within foreseeable future.

2.4 Virtual Machines

Virtual machines [Gol73] are other examples of sandboxes. Although they were were introduced several decades ago, in the 1960's by IBM to enable sharing of expensive mainframes, the concept is now more popular than ever and has revived in a multitude of computer system aspects that benefit from properties such as application mobility, increased security, co-existing operating systems, server consolidation, and utilization.

An early definition formalized virtual machines as software machines (VMM) with the following essential properties [PG74]:

- Equivalence: A program running under the VMM should exhibit a behavior essentially identical to that demonstrated when running on an equivalent machine directly.
- Resource control: The VMM must be in complete control of the virtualized re-

sources.

- Efficiency: A statistically dominant fraction of machine instructions must be executed without VMM intervention.

Today's virtual machines are divided in two groups: `system virtual machines` and `process virtual machines`. Xen [BDF+03], VMware [VMW99; VMW06], and VirtualBox [Wat08] are the most commonly known system virtual machines. Process virtual machines, best known examples being the Java Virtual Machine [LY99] and the MicroSoft Common Language Runtime [MWG], arose after the properties above were formalized and cannot fulfill the Efficiency property, so today, the first two properties are sufficient to characterize a virtual machine.

System virtual machines consist of a Virtual Machine Monitor (VMM), also known as a hyper-visor, that manages all instances of running virtual machines, and virtualizes a complete system environment, i.e. it shares the underlying physical hardware resources between each instance of a running virtual machine. The virtual machine instances execute a complete sandboxed operating system, known as the guest operating system. System virtual machines come in two forms depending on where the VMM is installed. In `Hosted Architecture`, the VMM is installed in an existing operating system, known as the host OS. This approach relies on the host OS to provide access to the hardware. In the other approach, `Bare-metal Architecture`, the VMM is installed directly on the hardware, thus including its own set of hardware drivers. The MiG-SSS is based on system virtual machines, so Chapter 4 elaborates on pros and cons of this type of virtual machine and brings examples of each architecture.

A process virtual machines, also known as an application virtual machine, only supports running a single application on top of an existing operating system. The virtual

machine process virtualizes an environment in which the guest process can execute. As shown in Figure 2.11, process level virtual machines are simpler because they only execute individual processes, each interfaced to the hardware resources through a virtual instruction set and an Application Binary Interface. Figure 2.11 illustrates the difference.

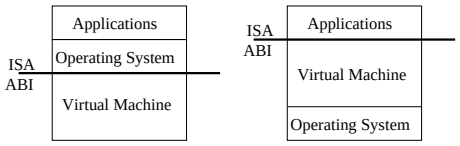


Figure 2.11: System VMs (left) and Process VMs (right)

The SciBy VM is a process virtual machine, and further details about this type of virtual machines are presented in Chapter 5.

2.4.1 Virtual Machines in a Grid Computing Context

Due to the renewed popularity of virtualization over the last few years, virtual machines are being developed for numerous purposes and therefore exist in many designs, each of them in many variants with individual characteristics. Despite the variety of designs, the underlying technology encompasses a number of properties beneficial for Grid Computing:

Platform Independence In a Grid context, where it is inherently intrinsic to move around application code as freely as application data, it is highly profitable to enable applications to be executed anywhere in the Grid. Virtual machines bridge the architectural boundaries of computational elements in a Grid by raising the level of abstraction of a computer system, thus providing a uniform way for applications to interact with

the system. Given a common virtual workspace environment, Grid users are provided with a compile-once-run-anywhere solution.

Furthermore, a running virtual machine is not tied to a specific physical resource; it can be suspended, migrated to another resource and resumed from where it was suspended.

Host Security To fully leverage the computational power of a Grid platform, security is just as important as application portability. Today, most Grid systems enforce security by means of user and resource authentication, a secure communication channel between them, and authorization in various forms. However, once access and authorization is granted, securing the host system from the application is left to the operating system.

Ideally, rather than handling the problems after system damage has occurred, harmful - intentional or not - Grid applications should not be able to compromise a Grid resource in the first place.

Virtual machines provide stronger security mechanisms than conventional operating systems, in that a malicious process running in an instance of a virtual machine is only capable of destroying the environment in which it runs, i.e. the virtual machine.

Application Security Conversely to disallowing host system damage, other processes, local or running in other virtualized environments, should not be able to compromise the integrity of the processes in the virtual machine.

System resources, for instance the CPU and memory, of a virtual machine are always mapped to underlying physical resources by the virtualization software. The real physical resources are then multiplexed between any number of virtualized systems, giving the impression to each of the systems that they have exclusive access to a dedi-

cated physical resource. Thus, Grid jobs running in a virtual machine are isolated from other Grid jobs running simultaneously in other virtual machines on the same host as well as possible local users of the resources.

Resource Management and Control Virtual machines enable increased flexibility for resource management and control in terms of resource usage and site administration. First of all, the middleware code necessary for interacting with the Grid can be incorporated in the virtual machine, thus relieving the resource owner from installing and managing the Grid software. Secondly, usage of physical resources like memory, disk, and CPU usage of a process is easily controlled with a virtual machine.

Performance As a virtual machine architecture interposes a software layer between the traditional hardware and software layers, in which a possibly different instruction set is implemented and translated to the underlying native instruction set, performance is typically lost during the translation phase. Despite of recent advances in new virtualization and translation techniques, and the introduction of hardware-assisted capabilities [AMD05; UNR+05], virtual machines usually introduce performance overhead and the goal remains achieving near-native performance only. The impact depends on system characteristics and the applications intended to run in the machine.

2.4.2 Security

Isolation is considered the primary characteristic of virtualization technology, and while the theory of confining an application in a sandboxed environment, from which it cannot escape, is attractive, the extent of isolation depends on the underlying virtualization technology. The general rule of thumb states that the isolation should be strong enough

to contain crashed guest applications or even crashed guest systems without affecting any other guest machine nor the host machine.

Process virtual machines such as the Java Virtual Machine or the Microsoft .Net VM are considered very secure due to their respective type-safe languages and the type-checking protection mechanism that verifies the bytecode before it is executed. Such type-systems can be proved sound [DE99; Sym99], and they are known for providing the most secure sandboxes.

System virtual machines are much more complex to implement since they virtualize an entire system, possibly all the way down to interrupts, I/O ports, and DMA channels. And currently, while process virtual machines have been in stable production for many years and have endured thorough tests, system virtual machines are evolving rapidly with more and more features. And with increased complexity and feature lists follow increased probability of software errors and thus possible threats. For instance, most virtual machines do not allow access to the host's file system, yet the VirtualBox virtual machine now features shared drives and shared clipboards between the guest and host system, thereby eschewing the isolation by providing a gateway for data to be transferred between cooperating guest VMs and the host system. Guidelines to virtual machine security can be found in a white-paper by cisecurity.org [CIS07].

Despite the regular occurrence of exploits found in system virtual machines*, no general technique for *attacking* system virtual machines has been found, only techniques to *detect* the existence of an underlying virtual machine [Fer06].

When deployed in a Grid context, virtual machines are only capable of isolating data and applications from other Grid users, and ensuring the integrity of the host system;

*For instance, a critical guest-to-host exploit was recently found in VMware fusion: `http://news.softpedia.com/news/Guest-to-Host-Exploit-Found-in-VMware-Fusion-109530.shtml`

there is no protection against the resource owner who, with a little effort, can gain access to the virtual machine.

Chapter 3

The MiG Remote File Access Library

The work presented in this chapter is derived from my Master's Thesis, in which a prototype of the Remote File Access library was developed. For this project, the library has been augmented and hardened through use by several standard applications. The full length papers [AV05; AV07] published on this library, can be found in appendix B.

3.1 Introduction

One consequence of allowing users to execute programs at remote sites, is that the applications and their data are not co-located. The problem of executing a program geographically separated from the files it accesses, is traditionally solved by staging all input files to the resource where the program is to execute, and vice verse for output files.

For more than a decade, the Grid has primarily been a domain for scientific applications, and many such applications only require access to a small subset of data from an exceedingly large data set. Often, only the first part or scattered fragments of input

files are really needed. While the conceptual simplicity of the upload/download model is evident and appealing, it entails many drawbacks for scientific applications:

- Job execution on the resource is delayed until all input files are downloaded. This drawback scales proportionally to the size of the files and results in wasting not only storage and network bandwidth, but also, and more importantly, lots of valuable time already allocated on the resource. The same issue holds true for output files. Only here, the finalization is delayed until all output files are uploaded.

- In situations where the application only accesses fragments of the files, this model results in excess data transfer and false storage requirements on the resource.

- Staging eludes streaming data back to the user's home catalog as the application runs. If this feature is viable, users can follow the progress of the application interactively.

- If job migration is supported, as in the *Checkpointing* feature of Condor [LLM88], a preempted job that is about to be resumed at another location, is forced to re-stage all input and output files.

Moreover, recent hardware trends clearly indicate that it is inefficient to copy data from place to place instead of just accessing them directly from where they originally resided. Moore's law has been remarkably accurate in forecasting increases and capabilities of hardware in computing technology. However, while most of the electronic aspects of computing technology have experienced exponential growth, mechanical devices like hard disks have been greatly outperformed.

Year	Network BW [Mbps]	Disk BW [Mbps]	Disk acc. time [ms]	Disk rot. speed [RPM]
1990	10	6	27	3500
1995	100	64	18	5400
1998	1000	80	16	5400
2002	10000	320	12	7200
2006		512	10	10000
Improvement p.a.	77%	32%	6%	7%

Table 3.1: Key numbers on network and disk evolution. The 100 Gb Ethernet standard was expected in 2006 but is delayed.

As shown in the logarithmic plot in Figure 3.1, network bandwidths clearly outperform hard disk bandwidths. Further, as shown in Table 3.1, key numbers on hard disk evolution are not very encouraging. The transfer rate is only one part of the disk speed, the other being the random-access time, which consists of the seek time and the rotational latency. While the transfer rate has improved over time, the random-access time is nearly constant due to mechanical constraints. The physical limitations on how fast the disk actuator arm can move to the desired cylinder, and how fast the disk spindle can rotate to get the desired sector under the read-write head, will always obstruct hard disk performance. Similarly, the network latency, which is the other part of the perceived network speed, also suffers from poor improvements, mainly affected by the impact of the finite speed of the medium, e.g. the finite speed of light. Over distance, the latency can be quite costly. For instance, on the 2.5 Gbps research network connection between Odense and Copenhagen (150 km), the latency is around 4 milliseconds. There are various ways to remedy this problem.

Consequently, although there are also ways of addressing disk latency problems, disk accesses are very time consuming operations, and performing them several times, as is the case when using the staging model, should be avoided.

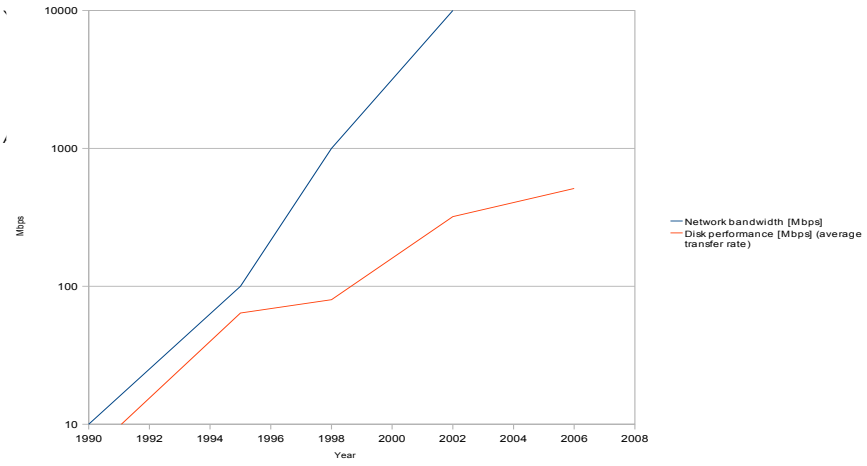


Figure 3.1: Evolution of network and hard disk bandwidths

In the remote access paradigm, applications are allowed to run in one location and access files remotely through the network, thus only paying the disk access penalty once. The remote access file service has several other advantages compared to staging techniques:

First off, it can enable the resource to limit its file retrieval to a set of file fragments holding the needed data. Secondly, *Latency hiding* can be applied by overlapping data transfer and computation. In the best case scenario, all transfer time is hidden, thus reducing the total wall clock time by the total file download time. Thirdly, using remote file access through an I/O library provides a higher-level specification of I/O operations that in turn permits greater flexibility as to how I/O is performed. For instance, a job may only need to access scattered fragments of the remote data sets. Also, one can choose to prefetch data according to re-definable access patterns. Other techniques are possible to add, such as encryption of all accessed data, support for custom file

transport protocols etc. Finally, intermediate results are available for the user to do real-time analysis before the job is finished at the resource.

The drawback in this paradigm is the more complex implementation and the fluctuating network performance due to varying latencies and bandwidths. Hence, the implementation must introduce a flexible mechanism to mitigate the impact of the network on the application.

3.2 Design

Providing on-demand transparent remote file access for a resource in a Grid environment first of all requires that certain file access routines on the resource are automatically overridden and redirected to the file on the server, described in 3.2.1. Next we need a protocol to transfer the data, 3.2.2. Third, since the local file management is overruled, a local file management mechanism to ensure correct file access behavior is needed, 3.2.3. Finally, the design of the caching mechanism and the prefetcher is explained in Section 3.2.4.

The main design challenge is to ensure that the on-demand remote file access can coexist with the demands of users and resources. Users should not be bothered with, or even aware of, the underlying complexities of the Grid, and should be able to submit their jobs to the Grid without rewriting or recompiling their applications.

Likewise, for resources to join the Grid, the administrative entry costs should be as low as possible and the security may not be compromised. Hence, all the functionality is placed in the Grid system and privileged access to the resources should be avoided. The decisions in the following sections reflect these observations.

3.2.1 Altering File Access Requests

Since a resource executes precompiled applications, it has to be extended with new functionality that implements transparent remote file access. Several approaches are applicable for this extension:

- Modify the OS source code directly
- Install a Remote File Access Server
- Use tracing facilities to intercept system calls
- A Statically linked library
- A Dynamically linked library

These approaches are explained in the following paragraphs.

Modify the OS Source Code: The most basic way of extending the behavior of certain system calls is to edit the kernel source files, i.e. implement the desired functionality in the file access routines directly, and recompile the kernel. A similar solution is to implement the functionality in a loadable kernel module and add it to the running kernel like it is done in NFS [PSB+00] and AFS [HKM+88].

Remote File Access Server: Installing a network server to provide the remote file access is a simple and well known solution and many such systems provide transparent access to remote resources.

Intercept System calls: In operating systems providing tracing facilities, a user process can attach to other processes owned by the user. The attach mechanism allows the process to intercept all requests to the kernel. Thus, the process can intercept file access routines and modify the behavior of the file if it is remote.

Statically Linked Library: Since file access routines are implemented as library functions in standard libraries, another approach is to replace these libraries with a new library implementing the desired functionality. Applications then need to be linked to this library.

Dynamically Linked Library: A similar approach is a dynamically linked library. Only, when using a dynamically linked library, applications need not be relinked, but are required to be compiled with dynamic linking.

The different approaches and their limitations concerning deployment in the Minimum intrusion Grid are summarized in table 3.2. The performance overhead, shown in the last column, is an estimate on the overhead incurred by the method of catching I/O calls in the different approaches.

In order for the file access model to comply to MiG, it must be ensured that it is entirely user-level and installable without administrator privileges. This excludes the first two approaches.

In the Ufo Global File System [MSS+97] it has been shown how to extend or alter the functionality of certain system calls by intercepting them in a user level module. The interception is achieved by standard tracing facilities supplied by most UNIX variants. This strategy avoids the need for recompiling, relinking and administrator privileges,

Approach	Requires root access	Requires recompiling	Apps. supported	Performance overhead
Modify OS Source	Yes	No	All	None
File Access Server	Yes	No	All	Medium
Intercept System Calls	No	No	All	High
Statically Linked Library	No	Yes	All	Low
Dynamically Linked Library	No	No	Dyn. linked	Low

Table 3.2: Approaches to transparent remote file access.

and grants transparent access to personal accounts at remote sites using different protocols. The only drawback to this model is the interception method, which is quite expensive. Intercepting system calls by means of tracing facilities is a great alternative for applications that issue a small number of system calls.

Since a statically linked library requires user applications to be relinked against the new library, this approach is not transparent for the user and thus conflicts with the third rule of MiG.

The best solution for our purpose is to implement the new functionality in a private set of the standard file access routines by means of a a dynamically linked library, and then preload this user-defined library in the application environment. In Linux, setting the LD_PRELOAD environment variable instructs the linker to first check the preloaded library for a matching symbol name. Thus, one can write a private set of `GLIBC` functions, while functions not overridden are automatically handled by `GLIBC`. This feature requires user applications to be dynamically linked, which most frequently is the case. Hence, statically linked applications are not supported by this model.

As an example of the idea, consider the following application that reads every other

chunk of 512 bytes from an input file. Assuming it is compiled by the user locally, the binary is then sent through the Grid to a resource that executes it:

```
int main() {
  int fd;
  char buf[512];

  fd = open(``inputfile'', O_CREAT|O_RDONLY,0666);
  while(read(fd, &buf, 512) != 0)
    lseek(fd, SEEK_CUR, 512);
  close(fd);
}
```

The proposed file access layer then automatically catches the `open`, `read`, `lseek`, and `close` calls and implements functionality that perform operations conforming to POSIX behaviour on the fragment of the remote file.

Thus, the library gives the user applications the illusion that job files exist in their entirety on the resource, yet they only exist on the server, and the server is only contacted when necessary.

Figure 3.2 shows the file access model. A set of file access routines issued from the user application are handled by the MiG file access layer that sends a request to the server (1). The server then replies (2), and an appropriate action is taken by the file access layer before the application receives the result (3). As indicated by the dotted lines, some requests can be handled directly, in case the requested data is already cached, and some need to be forwarded to the operating system.

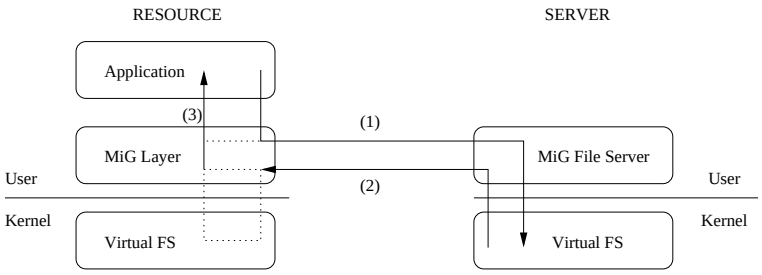


Figure 3.2: Overview of the MiG File Access Model

3.2.2 Data Transfer

Providing on-demand transparent remote file access for a resource in a Grid environment requires a secure, fast, and reliable data transport mechanism. Clearly, a protocol that only supports up- and download of entire files cannot be used for blocked remote on-demand file access. Hence, besides opening and closing a file, the protocol should support reading and writing ranges of randomly requested or written data.

The HTTP/1.1 protocol includes a byte range specifier in GET requests. This would suffice for read requests. However, since this specifier does not apply to PUT requests, the protocol would have to be augmented customarily for writing ranges of data.

GridFTP [ABB+01] is a high-performance, secure, robust data transfer mechanism specifically designed for grid environments. It is based on FTP with extensions for grid-specific requirements. Additional features such as parallel data transfer, striped data transfer, automatic or manual TCP buffer setting, and partial file transfer are included in GridFTP. However, since the protocol must be installed manually with root privileges, other alternatives are investigated.

Since no protocol supporting reading and writing partial files has been found to meet the requirements of MiG, a custom protocol based on the HTTP/1.1 is developed.

The protocol for transferring data between the resource and the file server is very

simple and should only support a range specifier and a small set of file-related commands, initiated by the resource:

- `open filename`: Establish a connection and open the file on the server. The server replies with the size of the file.
- `range filename offset length`: Read *length* bytes, starting at *offset* from the file. The server sends the requested range.
- `write filename offset length`: Prepare to receive *length* bytes and write them to the file at the given *offset*.
- `close filename`: Close the file on the server and close the connection.

File Server

In order to ease the implementation of fault tolerance, a stateless server is chosen in favor of a stateful server that loses all volatile state information in a crash. Hence, all requests are self-contained, and any backup server can respond to all requests.

Thus, if the primary file server crashes, the file access layer opens a connection to the secondary file server that opens the file, executes the request and returns the result back to the layer. To the application, this inconvenience is completely unnoticeable.

The drawbacks incurred by a stateless design, including longer request messages, slower processing due to the lack of in-core information to speed the processing, and the need to implement a low-level naming scheme to compensate for the target file translation, are considered less important than the issue of fault tolerance. In fact, messages are still very small and many requests, for instance `seek`, are handled locally on the resource. Furthermore, the low-level naming scheme only consists of mapping file

names to a socket and file descriptor. Finally, other methods to boost performance are into action, as described next.

Pseudo-code for a typical file transfer might look like this:

```
open network connection
open file
  while there is more data:
    read data from file to a buffer
    write data from buffer to destination
close file
close connection
```

Reading and writing would typically use the corresponding system calls or library functions built on top of them. Following the path of the data from disk to the network, data is copied several times. On read, it is copied from the disk to a kernel buffer and then from the kernel buffer to the application buffer.

Afterwards, on write, the data in the application buffer is copied to a kernel buffer and finally, it is transferred to the network card. First of all, the application buffer and the copying between the buffer and the kernel buffer are redundant and, secondly, all system calls are succeeded by a context switch between user and kernel mode, which quickly accumulates to be quite expensive.

The `sendfile` system call addresses this issue by eliminating the application buffer and allows for direct kernel-level copying from one file descriptor to another (including socket descriptors).

3.2.3 File Handling

In the remote file access paradigm, the local file management is overruled for all remote files that are being accessed. Hence, a mechanism to ensure correct file access conforming to POSIX-behavior is needed.

A shortcut to overriding the complete set of file manipulating routines is taken by actually creating the remote file on the resource. Besides the obvious advantage of not implementing all file access routines, including maintenance of their evolution, implementing complex UNIX functionalities is also avoided. Instead, the file is created on the resource and before any file access, it is ensured that the requested data is available.

As Figure 3.2 illustrates, some remote file accesses, i.e. `open`, upon server response result in requesting the resource kernel system to create the file. Other calls, i.e. `read`, remain in the MiG layer on receipt of the server response and returns data directly to the application, indicated by the dotted line. Had the requested data already been fetched, it would have been returned immediately without server or kernel intervention, indicated in the figure by following the dotted lines from the application through the MiG layer and back to the application. This issue is explained more detailed in Section 3.2.4.

Clearly, if the application continuously reads data that already has been read, or prefetched as explained below, and is available immediately from the cache, the performance of the application is improved significantly. If the proposed layer can return all requested data blocks immediately, thus bypassing the local kernel system, no system calls are needed to read the data. The kernel system is bypassed by mapping the file into memory, explained next.

Mapping of Files into Memory

The memory mapping design mimics the way user programs (ELF-format binaries) are loaded and executed in Linux, where the kernel's loader does not load the program into physical memory, but only sets up a mapping of the program in virtual memory. Pages of the binary are then mapped into regions of the virtual memory, and a given page is only loaded into physical memory when a page fault occurs due to the program trying to access the page.

Similarly, in the proposed model, all remote input files opened by the user application are created on the resource and mapped into memory, and individual pages will be loaded into the mapped image on demand when the application accesses them.

Although the file is initially empty, it is mapped in its full length. Accesses to empty regions of the file within the allowed scope then result in segmentation faults, which results in the requesting the pages from the server. Thus, the memory mapped image of the file can be highly fragmented, because only the needed data is retrieved.

Since access to an empty region within a file results in a segmentation fault, a procedure to handle segmentation faults must be introduced. This can be implemented using the `sigaction` system call that invokes a procedure upon receipt of a `SIGSEGV` signal. The procedure should then get the faulting address, determine the file owning the faulting address, translate the address into a file offset, and send a request to the server for the block surrounding the offset.

Mapping job-files into memory has several advantages:

- Simplified page administration: The layer only deals with memory addresses when accessing files.
- Direct memory access: The layer reads data from the socket directly into the

correct location of the mapped file image in memory. This prevents copying from a buffer.

- Enables kernel bypassing: Reading prefetched, cached or already copied data is returned immediately without a system call.

- Immediate support for user-mapped files: The memory mapped image may be returned directly to the application if it issues an `mmap` call.

It is important to stress that all job-files are mapped into memory, not only those that are explicitly mapped by the user application. As such, the memory mapped image of a file acts as the cache for the file in question. A `read` request would then copy the requested bytes from the cache, i.e. the mapped image, into the buffer supplied by the user; had the user mapped the file into memory explicitly, the file would be accessed by direct references to memory locations in the cache.

Figure 3.3 summarizes the design: The file is created and a dummy byte is written at the end of the file to expand the file to the size of the remote file. Then, the file is mapped into memory. The dummy byte at the end of the file ensures that the file size is the same as the remote file size, and all accesses to empty regions within the file image cause segmentation faults. The segmentation fault handler then sends a request for the missing data and copies it directly into the mapped file image at the address that caused the segmentation fault.

In the depicted scenario, the `read` call to read blocks 2, 3, and 4 is caught by the overridden `read` call, that just copies the blocks to the user buffer by a `memcpy`. Since block 3 is empty, a segmentation fault is raised on the start address of the first page in block 3, which enables the main page fetcher to get the block from the server. When data is transferred, the `memcpy` can complete, and the `read` call is executed.

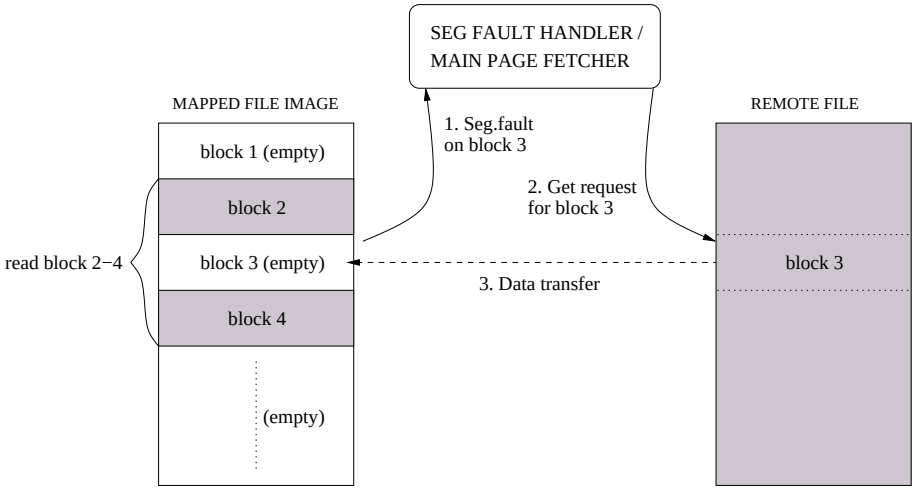


Figure 3.3: Data access design.

Virtual File Descriptors

File access routines are based on file descriptors that are dereferenced through the process file table into an internal kernel structure. The object of the file access layer is to emulate local file access by creating the file and fill in the requested pieces on-demand.

Therefore, a user-level structure is needed to handle all information, imposed by the file access layer, about the file. This structure is called a *Virtual File Descriptor* and contains extra information about a file, such as the length, the real file descriptor, the socket descriptor by which data can be retrieved, the current file pointer, the address at which the file is mapped in memory, etc.

Figure 3.4 shows how the virtual file descriptor interacts with the I/O subsystem in the kernel and keeps track of the mapped image of the file in memory. The figure shows a snapshot of the application execution, where 2 scattered fragments of the file have been accessed and therefore have been retrieved from the server.

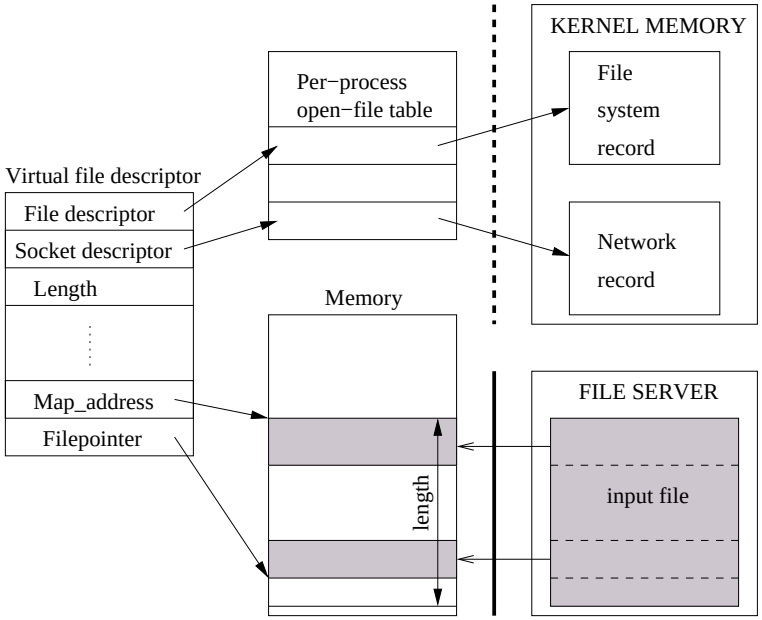


Figure 3.4: Interaction with the I/O subsystem

Overridden Calls and File Semantics

All functions that are overridden must exhibit the same behavior as the corresponding GLIBC function and conform to the ANSI C standard.

Thus, all functions are not only designed to have the same file semantics as the usual GLIBC functions, but must also carefully implement the same prototypes, error handling, etc.

As an example, Figure 3.5 shows how the internals of the library utilizes the virtual file descriptor and the memory-mapped image of the file to satisfy a `read` request from the user application.

First, the layer determines whether the file is a job-file or not. If not, the call is forwarded to GLIBC. In the other case, the virtual file descriptor for the file in question is retrieved. Then, the boundaries of the file are checked to avoid unnecessary network requests that do not get any data.

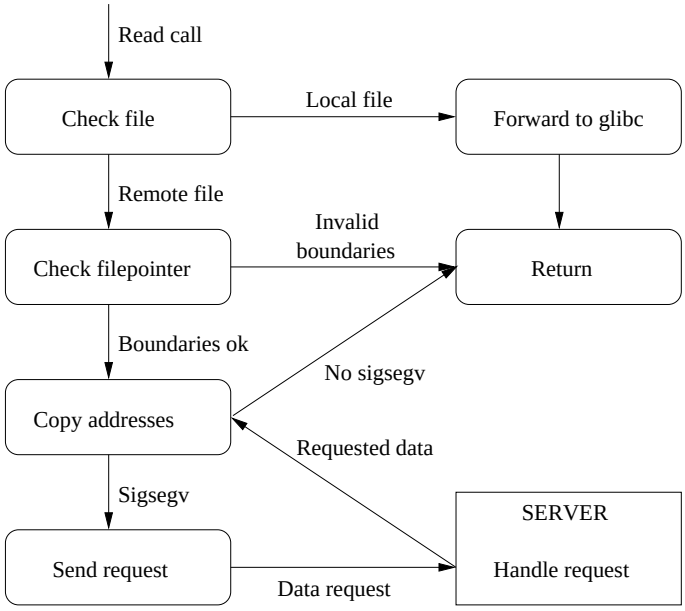


Figure 3.5: Control flow of a read operation

Next, the layer copies data directly from the file pointer to the user buffer, either a direct copy if all data is available immediately, or through a server request if the data is not available locally. If some of the requested data are not available in the mapped image of the file in memory, the copying may raise a segmentation fault. The segmentation fault is then caught, and a procedure to retrieve the requested data from the server is invoked.

Upon server response, the data are placed in the mapped file image, and the copying can proceed before the overridden `read` function returns.

The complete set of overridden calls are listed in table 3.3.

Opening, reading from, writing to, and closing a file results in contacting the server, while the remaining functions are handled locally. Seeking in the file repositions the file pointer in the virtual file descriptor.

Mapping a file is handled by simply returning the address of the mapped image,

Name	Description
open	Creates a connection to the server and opens the file.
close	Closes the file and closes the connection.
read	Reads a range of bytes from the file.
write	Writes a range of bytes to the file.
lseek	Repositions the file pointer in the file.
getc	Reads a byte from a stream.
putc	Puts a byte on a stream.
mmap	Maps a range of bytes from the file into memory.
munmap	Deletes a mapping for a range of bytes.
mremap	Expands or shrinks an existing memory mapping.
dup	Duplicates a file descriptor.
ftruncate	Truncates a file.
fopen	Opens a file and associates it with a stream.
fclose	Closes a stream.
fread	Reads from a stream.
fwrite	Writes to a stream.
fseek	Seeks in a stream.
ftell	Get position in stream.
rewind	Resets file position in stream.
fsetpos	Sets a new file position in stream.
fgetpos	Gets the current file position in stream.
fgetc	Reads a byte from a stream.
fputc	Puts a byte onto a stream.
fgets	Reads chars from a stream.
fputs	Writes chars onto a stream.
putw	Puts a word onto a stream.
getw	Gets a word from a stream.
ungetc	Pushes a char back to stream.
fscanf	Reads formatted input from a stream.
fprintf	Writes formatted output on an output stream.

Table 3.3: Overridden file access routines.

while unmapping just unlocks the pages. Effective unmapping is not applied, since we still keep a mapping of the file and unmap it ourselves when the file is closed.

3.2.4 Caching

Caching is introduced on the resource to ensure reasonable performance of the remote file service mechanism by means of reducing network traffic and disk I/O. If the data needed to satisfy a read request are not already cached, the request is delivered to the server. The server then performs the request, and returns the result back to the resource that caches the data. All accessed data is then retained in the cache, so that future accesses to the same data can be handled locally. In effect, caching decreases the server load and hence increases the potential for scalability.

The key challenge concerning performance in the remote access paradigm is to mitigate the impact of the network. In a Grid environment with varying network latencies and bandwidths, adaptation to the network is necessary when network resources fluctuate in performance as is the case in a dynamic, heterogeneous network environment.

Thus, the granularity of the transferred data is important to successful caching in terms of hit ratio and miss penalty. Also, prefetching can improve performance. These issues are discussed next.

Prefetching

Prefetching is a method of overlapping all kinds of I/O of a program with the computations of the program. The general idea is to instruct the input device to begin reading the next data after completing a read operation. When the program finishes operating on the first data item and requests the second, the input device has, ideally, already read

the second data item, which is ready for processing immediately. This method keeps the CPU and the input device busy and can improve performance significantly.

The basic caching scheme is augmented with prefetching in order to alleviate the network latency. All read requests are followed by a data request for new data, while the CPU processes the first read request.

Various prefetching algorithms exist:

- Sequential prefetching is the simplest kind of prefetching, in which subsequent block(s) are read and cached in the hope they will be accessed next. Clearly, this approach only works for applications exhibiting a sequential access pattern. Variants to this scheme are also applicable [MY01].

- Application specific algorithms focus on certain applications that exhibit the same access patterns, for instance matrix operations and searching in tree structures.

- Predictive algorithms focus on calculating the next block based on history and thus attempts to deduce future access patterns from past reference history [VL97].

The last two techniques are out of the scope for this work, and would require the user to specify a prefetching technique as part of the run-time environment. The sequential prefetching is the most generic approach, which makes it the best choice for a Grid environment.

The prefetcher is designed as a thread that succeeds each read request to the server with a request for the next consecutive block, as shown in Figure 3.6.

Before each read request, the prefetch buffer is checked to see, if the requested block is the one prefetched. Naturally, this approach only works for applications that use sequential access. However, the dynamic block size ensures, that non-sequential

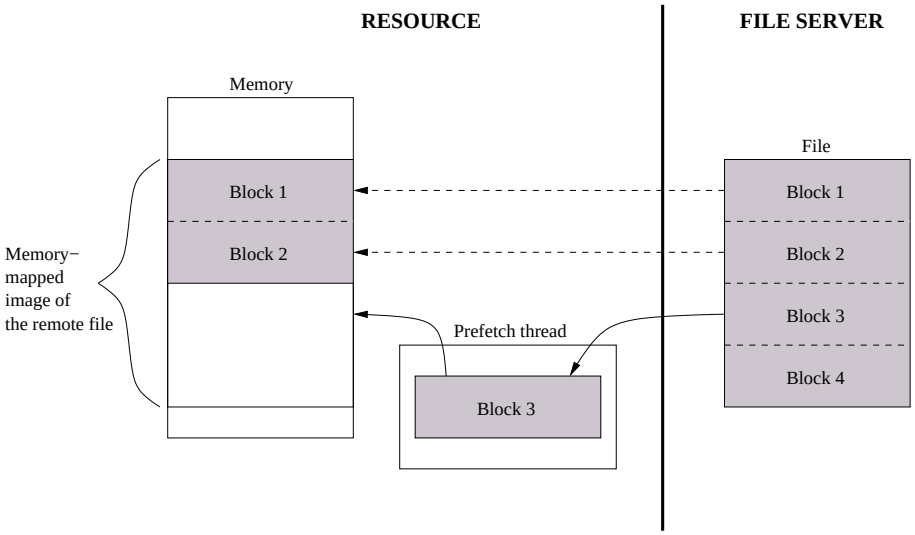


Figure 3.6: Prefetching using a prefetch buffer

access patterns are detected immediately, which enables the layer to lower the penalty incurred by a useless prefetching.

Every time a block is being read, it is noted whether the prefetching is finished and whether the prefetched block is the one we need now. Based on these observations, the block size either increases, decreases, or remains unchanged.

Block Size

The block size is important for several reasons, all depending on the user application. First of all, one can never tell in advance how many bytes a given application tries to read. As in the example code above, the user has chosen to read chunks of 512 bytes in each request. Clearly, sending a network request for 512 bytes is inefficient in a wide area network, and imaginably, this number could be anything.

Spatial locality is covered by caching more data than are needed to satisfy a single request [Den71; Smi82]. This increases the chance that several requests can be served

by the cached data. Increasing the block size increases the hit ratio, but the miss penalty also increases, because more data is transferred in the event of a miss.

Secondly, if the application reads all blocks sequentially, a large block size is preferable. The prefetcher would then be able to continuously read the next block, thus always reading a block of useful data ahead and hiding the latency. However, if the block size is too large, the prefetching would not finish before the data is needed for processing.

On the other hand, if the application only reads small scattered fragments of the file, reading ahead is useless and the application would have to wait for an excessively large block to arrive.

Since it is impossible to tell the nature of user applications in advance, it is a choice of either picking a general purpose block size, or changing it dynamically depending on the access pattern of the application. These approaches are discussed below.

Static Block Size A good choice of a static block size is often determined by the following two performance metrics:

- r_{max} is the *maximum achievable throughput* of the network connection, i.e. the asymptotic bandwidth of communication. It is obtained by transmitting very large messages.

- $n_{1/2}$ is the *half performance length*, the message length required to achieve half the asymptotic rate, i.e. the block size needed to achieve half that of the maximum achievable throughput. Put in another way, the *half performance length* is the message length, where one half of the transfer time is overhead waiting for the data to arrive, the other half is the effective data transfer time.

Figure 3.7 shows how to characterize a communication channel using these metrics and determine a general purpose block size. The pointed line indicates how to determine $n_{1/2}$.

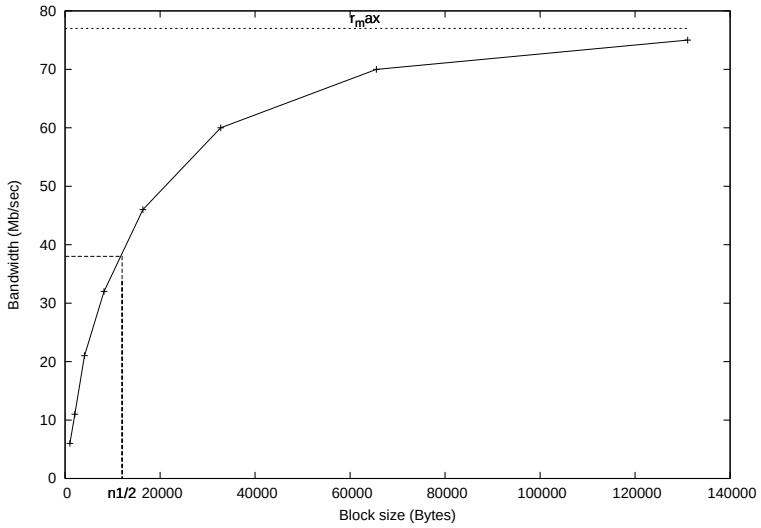


Figure 3.7: Bandwidth graph

The drawbacks of a static block size are, on the one hand the practical issues of determining $n_{1/2}$ of the communication between each resource and the corresponding file server, and on the other, the inability to cope with varying network conditions.

Dynamic Block Size In order to mitigate the penalties incurred by the network, dynamic adaptation to the continuously changing network conditions is required. A dynamically adapting block size is achieved by exploiting information about the network and the application. Before each read request the following is noted:

- if the preceding prefetching is finished or we have to wait for it, and
- if the prefetched block is the one that is needed now.

Denoting these events *pref_finished* and *correct_block*, respectively, the block size is adjusted between as follows:

```
if (!pref_finished && correct_block)
  #sequential access pattern:
  block_size <<= 1;
elif (!pref_finished && !correct_block)
  #non-sequential access pattern:
  block_size >>=1;
else
  ;
```

As soon as non-sequential access is detected, the block size is decreased if the process had to wait for the prefetcher to complete. Had the prefetcher already finished, then no time is lost due to wasted prefetching, which is why the block size then remains unchanged.

Similarly, if sequential access is detected, and the prefetcher finished in time, then the block size is adequate and remains unchanged. However, if the prefetcher did not finish in time, the block size is increased. The reason for this is to utilize the difference between varying latencies and block sizes. Increasing the block size increases the bandwidth with low extra latency overhead, thus increasing the chance for providing a large enough block for processing, such that the next block is finished in time.

The minimum block size is set to the size of a page in the local system. Keeping the block size in multiples of the page-size eases the management of the file in memory, and utilizes the local system most effectively, since the file is mapped into memory and

a `read` request results in copying the data from the server to the correct location in the memory mapped image of the file.

Thus, if the application tries to read bytes 1024 to 2047, the file access layer sends a request to the server for the block surrounding this range, i.e. if the block size is 8 kB, the layer sends a request for bytes 0 to 8095. All 8 kB are cached, but only the requested range within the block is copied to application buffer. A subsequent read call for any data within the range 0 and 8095 is then returned immediately.

Generally, all `read` requests result in the layer fetching a well-defined block, containing the requested range and an additional amount of data surrounding the range. The block is delimited by the pages surrounding the range.

Hence, as shown in Figure 3.8, if the user application requests bytes x to y, the layer fetches the range from the byte corresponding to the start address of the page containing x to the byte corresponding to the end address of the page containing y.

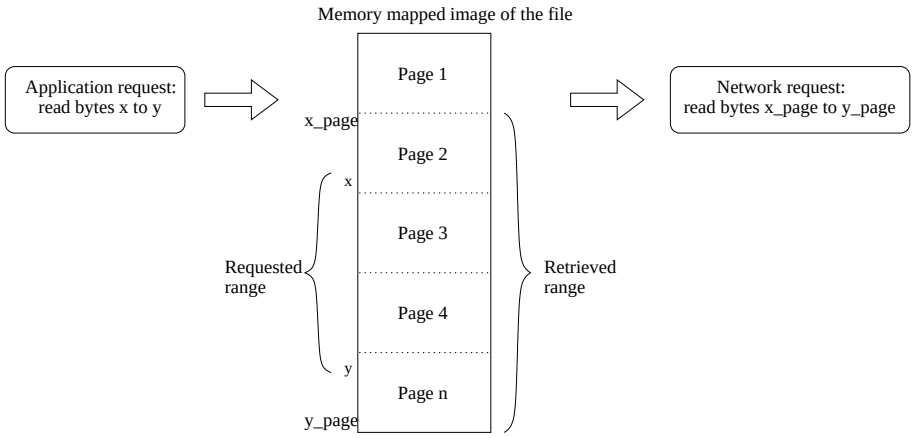


Figure 3.8: Translating a read request to a network request for a range of bytes.

3.3 Experiments & Results

The performance of the proposed model is measured by some experiments that cover the access patterns that apply to most applications. The results are then compared to, on the one hand, local execution, where the application and the files are co-located, and on the other, a standard Grid model where all files are downloaded prior to the job execution.

When comparing to a standard Grid model, we choose to compare to the performance of a basic transfer method, *cURL* [Ste]. This way any other project can perform the same experiment and by translation compare their performance to that of this model. Such systems could include bbftp [Far06] and UDT [Gu06].

The prefetcher and the dynamically adjusting block size are introduced in the model solely to improve performance. The impact of these mechanisms are studied lastly.

3.3.1 Experiments

The model is tested in 4 scenarios. Firstly, we investigate the basic overhead, secondly the performance in an I/O limited application is examined. Thirdly an I/O balanced application is analyzed, and finally we test the model in a scenario where only a small portion of a huge file is used.

In order to determine the basic overhead of the remote access protocol, the first experiment simply reads a 1-byte file and verifies the content. This experiment provides us with a baseline for the performance of the remote access model.

In the second test, the application checksums a 1 GB file. The calculations that are involved are so few and simple that the application is simply limited by I/O performance. The latency of getting a new page from disk is less than retrieving the data

through the remote access layer. Thus, this is the kind of application that does not really perform well on Grid. Still, the copy-semantics should be faster than the remote access layer, since bulk transfer of a complete file is more efficient than the blocked access model.

In the third experiment, a 1 GB input file, which requires some processing from the application is traversed. The input file contains a series of numbers that the application reads and then computes a corresponding fibonacci value on one of the numbers. This test will show the benefit of using prefetching in combination with starting the job immediately without waiting for the input file to arrive and should prove favorable to the remote access model.

Finally, a large file containing a B+ tree of order 4 is searched using a random key. The test is run with 10 different keys. A B+ tree is an ideal structure for the remote access model since it is designed to branch out in a large number of directions and to contain a lot of keys in each node. This ensures that the height of the tree is relatively small. Thus, only a small number of nodes, one in each level of the tree, must be read to retrieve an item.

3.3.2 Results

Table 3.3.2 shows the results of the 4 experiments. The results of using the proposed layer, shown in column 4, are compared to a model that just downloads the entire input file and executes the job, column 3. Local execution with local input files is shown in column 2. All experiments are performed on the same dedicated resource and server, both 2.4 GHz Intel Celeron with 512 MB RAM, using a 100 Mbps network connection in a LAN.

Experiment	Local	Copy	Remote
Overhead	0.0002	0.1520	0.0080
I/O-bound	50.1100	130.1000	114.0300
Balanced	632.8300	721.2200	600.7200
Partial access	0.0002	30.6920	0.0186

Table 3.4: Result of experiments in seconds.

As the result of reading a 1 byte file reveals, using the proposed remote file access layer does not incur much overhead compared to the local execution. This experiment also shows that the layer is faster than using external *cURL* to download the file.

The result of the I/O intensive application is surprising since bulk transfer should be faster than requesting the entire file block-wise. The reason why the remote file access is faster is due to a combination of the prefetcher and the server. The prefetcher keeps increasing the block size, because it detects sequential access, and the server is actually faster than the dedicated apache server that *cURL* consults.

The result of the balanced application, the Fibonacci experiment, shows that the layer is able to hide the transfer time during data processing. Amazingly, running this application using the proposed layer is faster than local execution. This is due to a combination of perfect prefetching and the design of the remote file access model, shown in figure 3.2. Almost all read requests are returned immediately by the file access layer without server or kernel intervention. Thus, all requested blocks are already in memory prior to the read calls. During local execution, all blocks are fetched using a system call and expensive disk access.

The B+ tree experiment performs excellently on the proposed model. The depth of the tree is 9, hence only 9 blocks are fetched plus an additional header block and wasted blocks from the prefetcher. Often the size of the B+ tree file is much larger than this

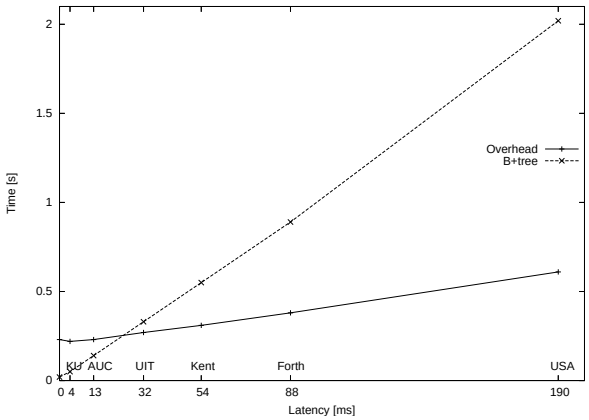


Figure 3.9: Results of the overhead and B+tree applications

one (357 MB), which in effect prohibits this kind of applications from execution with Grid implementations that use the download model. The speedup between the download model and the proposed model is a factor of 1650. Naturally, a native execution baseline is much more efficient at this stage, however overheads less than a second should be acceptable for any job one may choose to submit to Grid.

3.3.3 Performance in a heterogeneous network

The ability to mitigate the penalties incurred by heterogeneous network conditions is best illustrated by the following figures that show the impact on the execution time as the latency between the resource and the server increases. The latencies to different university centers are simulated on the file server by inserting a sleep call between the request and the response.

Figure 3.9 shows the overhead and B+tree applications. Since the overhead application only reads 1 byte, the effects of increased latency are noticeable but insignificant.

Since the B+ tree application reads a fixed number of scattered blocks from the file, the latency is added to each block transfer time.

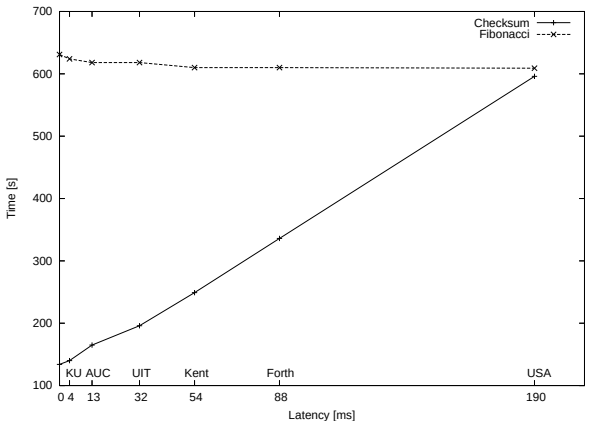


Figure 3.10: Results of the checksum and fibonacci applications

In Figure 3.10 the benefit of a dynamic block size is apparent: The running time of the fibonacci experiment is unaffected by the increased latency. As the latency increases, the library fetches larger blocks, thus achieving a larger bandwidth and providing more data for processing in each request.

The checksum application is more affected by the increased latency, because the application is I/O bound.

3.4 Conclusion

The on-demand transparent remote file access layer presented in this chapter is specifically designed towards the MiG project without compromising neither the design requirements nor the features of this Grid project.

Using this user-level layer, it has been shown that a resource does not need to neither download entire input files before job execution nor upload output files after the job has terminated. Instead it just starts the job, downloads portions of the input file when needed, and uploads modified data when written. This functionality is achieved without

requiring user application to use special API's or to be recompiled.

The results show that this model provides access to the Grid for a whole niche of applications that were previously impeded by unnecessary transfer of an enormous amount of data, namely applications using partial file traversal on huge input files, such as B+ trees.

This is also the case for applications such as high-energy physics applications that analyze relatively small fragments of massive physics database files.

Chapter 4

The MiG Screen Saver Sandbox

Many people are willing to donate computing power from their personal computers when they are not in use and just sitting idle in screen saver mode. The MiG Screen Saver Sandbox, or MiG-SSS, is a client software package that runs on standard desktop Linux and Windows computers and harnesses the excess compute power, whenever the screen saver is active. Since host system is an all-important topic when using public machines for arbitrary applications, the applications are enclosed by a sandbox environment, subject to strict security constraints. Without virtual machines to provide a sandbox environment, experience shows that people are, with good reason, reluctant to put their resources on a Grid where they have to not only install and manage a software code base, but also allow native execution of unknown and untrusted programs. All these issues can be eliminated by introducing virtual machines.

The system has been available for download since mid 2006 and several research projects from the fields of molecular chemistry and nano-science have been deployed on the resource farm. Approximately 1000 people have downloaded the sandbox.

This chapter explores the potential for using these machines, all the requirements on

the sandbox, how screen saver resources differ from normal resources, how to integrate them in the MiG middleware, and how to facilitate general usage of the resources. The full length paper published on the MiG-SSS [AV06] can be found in Chapter A.

4.1 Introduction

Cycle scavenging, or Screen Saver Science, is an increasingly popular computing paradigm used within many fields of science that seek to tap the enourmous amount of unused processing power from the millions of Internet-connected computers around the globe. The paradigm is best known from the many successful @home projects, such as SETI@home [ACK+02] and Folding@home [LSSP09], where standard desktop computers from homes and offices contribute to scientific simulations and analysis, whenever they are in screen saver mode. These systems have demonstrated a unique and huge public involvement which has led to an unprecedented amount of aggregated computing power. However, these @Home systems belong to another type of distributed computing, namely Public Resource Computing, or Volunteer Computing, which differ from Grid Computing systems in that the client software running behind the screen saver is an embedded dedicated application that continuously requests new *data* to analyze. In Grid Computing, the resources should be capable of performing any type of computation, thus continuously requesting new *applications* to execute.

One of the Grid incentives is to make it possible to share and effectively use distributed resources on an unprecedented scale. Specifically, this includes harnessing the unused capacity of idle desktop PCs. A lot of research has been done to Grid-enable a novel set of computer architectures, yet no widely accepted system to effectively scavenge idle cycles, in particular idle Windows cycles, has been found.

The purpose of MiG-SSS is to scavenge idle desktops for general scientific use. First of all, a method to gain access to the CPU cycles on the idle resource must be found. To this aspect, host system security is a major issue. Ideally, a resource owner should neither install any software nor execute any foreign applications that, intentional or not, could compromise his system. Secondly, the resouce, most likely hidden by a Network Address Translation router, must be attached to the Grid. Thirdly, it must be ensured that a given resource has installed the correct software base that a given Grid job requires. Finally, to improve things further, extra features relevant to the Minimum intrusion Grid are discussed.

4.1.1 Related Work

A few projects have been found to combine Screen Saver Science with Grid computing, for instance the Entropia Virtual Machine [CCWY05], which is a commercial product (company ceased their operations in 2004), and [FDF03] that is merely an extensive introduction to the approach of using virtual machines for Grid computing. BOINC [And04] is a software platform that allows many different distributed computing projects to utilize idle volunteered computer resources. Many Public Resource Computing systems use BOINC and research groups can create new projects. A project involves a set of applications that will be run in a BOINC client on a user's resource. As such, BOINC could be used for this project by running the proposed virtual machine as the project application. However, as noted in the introduction, the work load and the financial expenses necessary for setting up a new project are too high for small projects.

The Screen Saver Science project [GS08] uses the Java Virtual Machine (JVM) as a sandbox, thus taking advantage of the security provided by the JVM and the use

of a type-safe programming language, and the portability of the Java bytecode. The bytecode greatly simplifies development for a heterogeneous architecture. Using the Jini [AGG02] network technology and other high level packages, it allows for communication between the clients, thus supporting distributed applications. There are several drawbacks to this solution: Most significantly, as the authors note, Java is not designed for the application area, and there are no security mechanisms in Jini, so security is solely based on the JVM. Cyclone [JPMC00] is a similar system based on the same technologies. None of the systems seem to be in deployment.

4.2 Design

The basic idea is to let resource owners install a sandbox to provide a secure execution environment in which the Grid job is completely isolated from the host machine and vice-versa. Such a sandbox can take the shape of a system virtual machine, which is the approach we have taken in this work. A *system virtual machine*, as opposed to a *process virtual machine*, provides a complete system environment, which should be much simpler to grid-enable and get up and running with an embedded scientific execution environment.

Two techniques can be used to provide a system virtual machine: Emulation of a non-native system and Full virtualization of a native system. Emulation provides the functionality of the target processor completely in software, which makes it a very secure approach. Also, the ability to emulate one processor on top of any other processor makes it ideal for this scenario. However, the method of interpreting the entire guest operating system, rather than running it on the native hardware, results in a significant performance drawback.

Generally there are two types of native system virtual machine architectures: Hosted Architecture, and Bare-metal Architecture, see Figure 4.1. On a basic level, the difference between these types are their implications on determinism when using real-time operating systems, I/O accesses, and their ease of use. As the two first factors are not important for this project, focus will be on how the resource owner is affected by installing the virtual machine.

The Bare-metal Architecture technique is the classic approach to system virtual machines as originally proposed[PG74]. The Virtual Machine Monitor, the software layer that virtualizes the hardware, is installed directly on the hardware and all guest operating systems are then installed on top of the VMM. Thus, the VMM needs to be running in the highest privileged mode in x86 architectures, the `Ring 0`, while the guest systems run in lesser privileged modes, `Ring 1` or `Ring 3`. The main advantage of this approach is that the VMM transparently can intercept all interactions between the guest systems and the hardware resources. Therefore, the Bare-metal architecture is the most efficient virtualization technique. Xen [BDF+03] is a very popular virtual machine using the this approach. However, its disadvantage disqualifies it from being used for Volunteer Computing: Requiring users to explicitly port their operating system to run in a lesser privileged mode and interposing the VMM layer between the OS and the hardware, thus wiping out their existing operating system and replacing it with the VMM, is obviously too intrusive.

The Hosted Architecture approach is not as efficient as the Bare-metal approach, but the installation is similar to installing any other type of application. As shown in Figure 4.1, the virtual machine is installed on top of an already installed host OS, thus creating a secure sandbox that allows an application written for one operating system, e.g. Linux, to be executed in another, e.g. Windows. Common virtual machines using the hosted

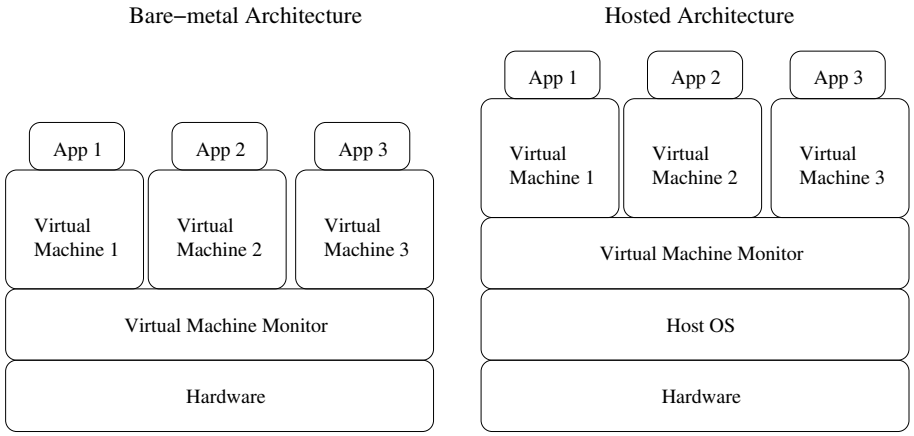


Figure 4.1: Full Virtualization

architecture technique include VMWare Workstation [VMW99] and Parallels [Par06].

The performance penalties incurred by the extra software layer in the hosted architecture are mitigated by having host OS kernel support that enables the guest OS to run most of the target application code directly on the host processor, thus achieving near native speed.

Regarding security in the context of host system integrity, the virtual machine is a user space process that cannot do any harm to the host system as long as the permanent storage is protected properly. If the virtual machine is destroyed by a malicious application, the host system is not affected, and the virtual machine can start afresh.

The following such solutions exist as free products for the Windows platform (as of 2006):

- VirtualPC [PC06] (Linux is officially not supported as guest OS)
- VMWare Player [VMW06]
- Qemu [Bel05] + Qemu Accelerator Module (A compatibility layer to avoid full emulation)

Having downloaded and installed one of the virtual machines, a resource owner only needs to install a screen saver that starts the virtual machine upon activation and a tailor-made Linux image that is capable of running the Grid resource software automatically. In this manner, when the resource goes into screen saver mode, the virtual machine is activated and the Linux guest operating system is booted.

The main problem with scavenging personal computers is that the vast majority are hidden behind a NAT router, i.e. they do not have global IP address and are therefore not reachable from the Internet. Hence, to enable the sandbox for Grid Computing, care must be taken to circumvent the missing inbound Internet access. Naturally, this issue is highly dependent on the Grid middleware in question, but as MiG uses a push model where communication is initiated by the server, some adjustments had to be made specifically for sandboxes. The sandboxes thus become a special type of resource that requires special attention from the Grid scheduler.

4.2.1 The MiG Linux Image

As explained above, all that is required for a resource to join MiG, is to craete a grid user account and support for incoming SSH and outgoing HTTPS.

So basically, the MiG Linux image can be built using any Linux distribution that runs an x86 system. Since the virtual machine provides a standardized virtualized set of hardware, compatibility amongst the wide range of different hardware setups on the resources will not be an issue.

The main concerns with respect to the distribution is the size and the start-up time. Both issues matter only for practical reasons: The size should be minimized to avoid an excessively large download, and, naturally, the start-up time should be minimized as

much as possible.

4.2.2 Runtime Environments

Once the basic sandbox is in place it is possible to execute user applications within the virtual Linux machine. Many applications can be passed as executables from the Grid job and these need no further components to execute. Other, commonly used, applications may benefit from a preinstalled runtime environment, such as they are found on ordinary Grid resources.

Installing runtime-environments in the sandboxed environment could be done as on a conventional resource, which however would require the PC owners to personally maintain the sandboxed Linux distribution and this model is thus not desirable. Alternatively the sandbox image could be distributed with the initial Linux image, but this would greatly increase the size of the distribution image and in addition be a very static model.

The chosen solution allows individual research groups to maintain runtime environments for the sandboxed resources and at the same time allow the individual PC owners to control which runtime environments are downloaded. The runtime environments are kept in individual virtual disk-partition, in the form of a single file.

The PC owner can download individual runtime environments from the VGrids, MiG's notion of a virtual organization [KV06], that maintain the runtime environments and when a job that uses a runtime environment is received by the sandboxed resource, the virtual Linux machine will mount the file system that contains the runtime environments. This way each runtime environments is kept isolated from the rest of the system, and can easily be build and maintained by the research groups that need them

to be available for their executions.

4.2.3 Scheduling

Existing Screen Saver Science systems all target problems that have many, usually millions, of independent tasks that often run for tens or hundreds of hours. Once a task has been assigned to a computer it will be processed while the computer is in screen saver mode. Processing is suspended if the screensaver is suspended and similarly resumed again along with the screensaver. An artefact of this model is that one never knows when the result of a given task is ready, and it is very hard to determine if the task has been lost or if it is simply only allowed to proceed very slowly.

For applications such as computational chemistry this model is very poorly suited. The number of tasks is usually in the tens or hundreds and it is often the case that analysis of the results can only start once the result of every task is in.

This is easily addressed by putting an upper time limit on each job, and if the time limit is exceeded, the job is resubmitted to another resource. However, in order to schedule a job with a deadline to a screen saver resource, we need to know how long the resource is availabe, i.e. how much time it takes before the screen saver is deactivated. To predict the available time slot of a screen saver resource, we use exponential average on an hourly basis, which has proved to converge against the actual resource idle time quite fast.

4.3 Implementation

To deploy the MiG sandboxes, the sandbox image had to built, MiG-specific sandbox resource files had to be embedded in the image, a screen saver for Windows and a

daemon for Linux computers were necessary, and finally, a web page with a facility for monitoring, and an easy-to-use download package including installer files had to be provided.

4.3.1 Sandbox Image, Version 0.x

As a base for the MiG sandbox image the `ttylinux`[Sch01] was chosen for the first version. It is a minimalistic distribution that is easy to customize and, despite its very scarce space usage, provides an environment similar to larger distributions. The only shortcoming is the missing HTTPS support, so this had to installed manually.

The generic MiG Linux Image consisted of a kernel and a RAM-disk that altogether take up less than 3 MB. An online generator modifies the generic image by giving it a unique resource name and a session id needed for requesting a job. Further, the resource owner can choose the size of a hard disk image file to provide as storage for the sandbox. Thus, certified resource owners can have a complete image built with a unique key allowing the sandbox to automatically request and execute Grid jobs.

In the standard MiG model, the identity of a resource requesting a job is verified by keeping the public SSH key of the resource in the MiG system and copying all job files to the resource over SSH. The sandbox model however, is modified to use a pull model on the resource where all files are transferred using HTTPS. Hence, a firewall in front of a resource only needs to be open for HTTP and HTTPS to allow the resource to run Grid jobs.

4.3.2 Sandbox Image, Version 1.x

After having deployed a few research projects, it turned out that `ttylinux` was too minimalistic; constantly building new software packages required by the users, typically in the form of libraries, was too cumbersome, and as time progressed, it became difficult to find versions compatible with the kernel system. Eventually, we had to change to a new system with a more advanced package management and update facilities. The system is now based on Gentoo Linux, so the image now takes up approximately 50 MB.

For the application developers who wish to test that their applications behave correctly in the sandbox, there is a development image available, which one can start up in any x86 virtual machine.

4.3.3 MiG-specific Resource Files

There are only 2 scripts necessary for a resource to act as a MiG resource: The *frontend_script.sh* and *masternode_script.sh*. Normally, these scripts are initially pushed out to a resource trying to register, but in this case, the scripts are requested from the MiG-servers during boot time. The division in two scripts is made to support cluster setups, where only one machine acts as a front-end, and all the worker nodes get jobs from the front-end and execute them in the *masternode_script.sh*. There are only minor modifications to the *frontend_script.sh* in order to support sandboxes, while the *masternode_script.sh* is unmodified.

4.3.4 Uniqueness and Identification

Every sandbox is equipped with a unique key which it uses to request jobs. This way, the MiG servers can identify a sandbox resource, since the key is kept secret between the resource and the MiG servers. The key is root-protected in the *frontend_script.sh* to ensure that jobs executed the user-space *masternode_script.sh* do not alter the key. In addition to the secret key, a session-ID is used to identify jobs submitted to the resource, and to control the access rights of the job.

4.3.5 Screen Saver Interaction

To get MiG-SSS into production, a screen saver, an installer, and a Linux daemon was developed and made available over the MiG website. People who download the sandbox then get a package consisting of the sandbox image, a hard disk image, and an installer that installs the screen saver and the virtual machine. This is fairly straightforward, yet extremely important for a successful product. The process of wrapping everything up nicely in cellophane is time-consuming but cannot be underestimated. For the Windows version, a screen saver was implemented in Python using the `pyscr` and `ctypes` modules. The following code illustrates the usage:

```
class MySaver(pyscr.Screensaver):
    def initialize(self):
        #called once when the screensaver is started
        reg_handle = _winreg.OpenKey(
                    _winreg.HKEY_CURRENT_USER,
                    "Software\\MiG-SSS\\")
        value, stype = _winreg.QueryValueEx(reg_handle, "")
```

```
        MiG_dir = value
        #go to MiG dir and initiate VM
        os.chdir(MiG_dir)
        cmd = "qemu -L pc-bios -boot d
                    -cdrom MiG.iso -hda hda.img
                    -kernel-kqemu"
        #launch the virtual machine
        self.process = subprocess.Popen(cmd)
        #launch a screen saver
        self.win_ss_process = subprocess.Popen(
                    MiG_dir+os.sep+"ssstars.scr /s")

    def finalize(self):
        #called when the screensaver terminates, kill VM
        ctypes.windll.kernel32.TerminateProcess(
                    int(self.process._handle), -1)
        ctypes.windll.kernel32.TerminateProcess(
                    int(self.win_ss_process._handle), -1)
```

An installer was built using the NSIS win32 installer system. Apart from copying needed sandbox files and system files to the host computer, it creates an entry in the Windows registry which is used for noting the installation directory. As shown in the screen saver code above, the screen saver then launches the virtual machine from this location.

For the Linux version, a screen saver wrapper, written in Python and based on the `xscreensaver` utility, is attached to the download package. Its functionality is similar to the Windows version above.

4.4 Experiments & Results

To verify the model before the first release, 8 sandbox resources were connected to MiG and 25 jobs that we pointed out as sandbox jobs were submitted. The 8 resources are all identical Windows PCs that host the virtual Linux machine that runs the Grid client code and includes the required runtime environment. The jobs are all NAMD [PBW+05] jobs, which is a software package for simulation of bio-molecular systems.

Since we had 8 machines, we chose to submit 25 jobs. Thus, there were 3 jobs for each machine and to avoid balanced execution, one last machine needs to execute one additional job before the experiment is completed. Further, using the 'minimization' option in NAMD effectively ensures different running times, thus ensuring unbalanced execution.

Running the jobs sequentially on one of the PCs results in a total running time of 2 hours, 25 minutes, and 22 seconds. When submitted to the Grid, the 25 jobs completed in 31 minutes and 45 seconds.

4.5 Conclusion

This work has shown how to eliminate the factors that have previously impeded the fusion of Public Resource Computing and Grid Computing to effectively utilize idle CPU cycles from desktop machines for any kind of Grid job.

The prohibiting factors include NAT-hidden resources, means to utilize Windows desktops, the workload required by a non-expert resource owner to install and manage all resource software, and the security issues involved with installing a large software base on the resource.

Using sandboxing technology and a generic Linux image, the Minimum intrusion Grid has successfully eliminated all of these limitations. Users need only download a bundle consisting of a screen saver, a virtual machine, and a special MiG Linux image in order to share their idle resources, whether they run Linux or Windows. The MiG system has proved flexible enough to easily deal with computers behind network address translators, and mobile processes and automatic resubmission of jobs solve the problem with resources that are cut off the network or leave the screen saver mode. Finally, the sandboxed environment ensures that the host system cannot be compromised.

Using this approach, a desktop computer volunteers as a Grid resource upon screen saver activation, and as soon as the screen saver is deactivated, the executing job either stops or migrates. Thus, the resource owner is completely unaffected by the Grid job.

Chapter 5

The Scientific Bytecode Virtual Machine

Although the MiG-SSS could close the gap between Volunteer Computing and Grid Computing by enabling easy access for typical desktop computers, it soon ran into limitations. On the positive side, the system virtual machines on which it was based kept improving, most notably on the performance side, where they reached near-native speed with hardware-assisted virtualization technologies and new techniques to address x86 virtualization problems [AA06; RI00]. However, as detailed in the introduction, personal desktop computers were outstandingly overhauled in the segment for personal devices by new powerful and radically different architectures. Today, the GPUs from NVIDIA and ATI and the Playstation 3s account for more than 20 times as many TFLOPS than PCs do in the Folding@Home project.

The overall problem with system virtual machines is the implementation complexity in developing a machine for every platform type, each capable of emulating an entire hardware environment for essentially all types of software. While work is on the way for

system virtual machines to utilize GPUs, the process of porting system virtual machines to new architectures is a significant task.

Since the application domain in focus is scientific applications only, there is really no need for full-featured operating systems. Process level virtual machines are simpler and much more portable because they only execute individual processes, each interfaced to the hardware resources through a virtual instruction set and an Application Binary Interface.

Using the process level virtual machine approach, the virtual machine is designed in accordance with a software development framework. Developing a virtual machine for which there is no corresponding underlying real machine may sound counterintuitive, but this approach has proved successful in several cases, best demonstrated by the power and usefulness of the Java Virtual Machine [LY99]. Tailored to the Java programming language, it has provided a platform independent computing environment for many application domains, yet there is no commonly used real Java machine *. The Full length paper on the machine presented in thischapter [AV08] can be found in Appendix D. Another paper, located in Appendix E, has been submitted to Journal of Grid Computing, 2009.

5.1 Introduction

Many virtual machines exist and many of them have been combined with grid computing. However, most of these were designed for other purposes and suffer from a few problems when it comes to running high performance scientific applications on a heterogeneous computing platform. Grid computing is tightly bonded to eScience,

*The Java VM has been implemented in hardware in the picoJava core [MO98]

and while standard jobs may run perfectly and satisfactory in existing virtual machines, 'gridified' eScience jobs are better suited for a dedicated virtual machine in terms of performance.

This approach addresses these problems by developing a portable virtual machine specifically designed for scientific applications: The Scientific Bytecode Virtual Machine (SciBy VM).

The machine entails a virtual CPU capable of executing platform independent bytecodes corresponding to a very large instruction set. An important feature to achieve performance is the use of optimized native libraries for the most prevalent algorithms in scientific applications. Security is obviously very important for resource owners. To this end, virtualization provides the necessary isolation from the host system, and several aspects that have made other virtual machines vulnerable have been left out. For instance, the SciBy VM supports neither system calls nor I/O.

Similar to Java, applications for the SciBy VM are compiled into a platform independent bytecode which can be executed on any device equipped with the virtual machine. However, applications are not tied to a specific programming language. As noted earlier, researchers should not be forced to rewrite their applications in order to use the virtual machine. Standard ANSI C compilers accept a broad range of programming languages typically used in scientific environments, and they are built in a modular fashion, where only the back end needs to be modified in order to support a new architecture. Thus, by providing a compiler for the SciBy VM, we get the scenario depicted in Figure 5.1, where any type of device can execute the application.

To evaluate the machine, we demonstrate several use-case scenarios from some of the intended application domains. Further, we show the ease of porting the machine and distributing its jobs to a variety of predominant architectures and compare the results

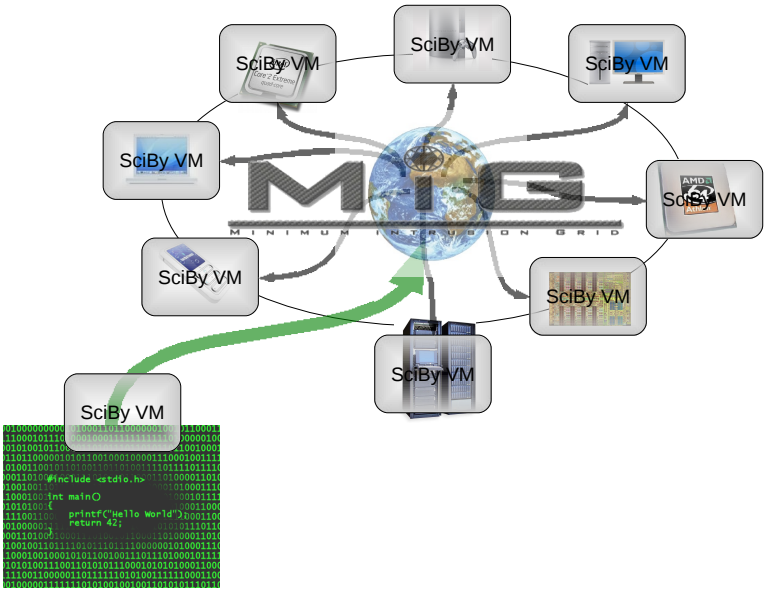


Figure 5.1: The SciBy VM can provide uniform access to many types of hardware deployed in a grid.

with native execution.

5.1.1 Enabling Limitations

There are a some important limitations that greatly simplify the development of the SciBy VM. Firstly, the implementation burden is lessened drastically by only giving support for running a single sequential application. Giving support for entire operating systems is much more complex in that it must support multiple users in a multi-process environment, and hardware resources such as networking, I/O, the graphics processor, and 'multimedia' components of currently used standard CPUs are also typically virtualized. In addition to all physical devices, a virtual machine can even be equipped with more virtual devices than those that are available to the underlying system. All devices are emulated by the VMM and virtualized as well-known standard hardware devices for

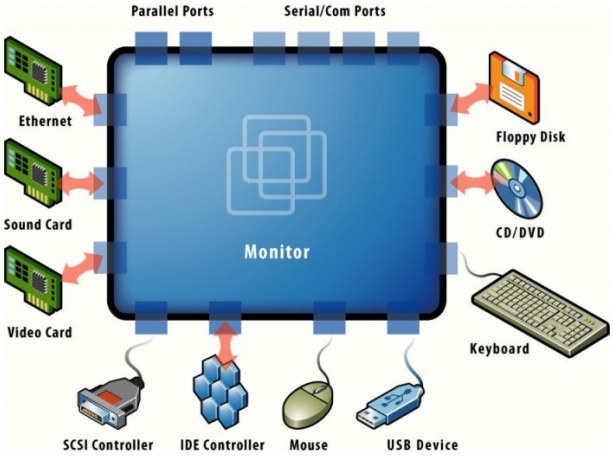


Figure 5.2: Device and I/O virtualization. Copyright VMWare

each instance of virtual machine. As illustrated in 5.2, such devices are numerous.

Finally, virtual machines allow fine-grained control over the actions taken by the code running in the machine. As mentioned in Section 5.4, many projects use sandbox mechanisms in which they by various means check all system instructions. The much simpler approach taken in this project is to simply disallow system calls. The rationale for this decision is that:

- scientific applications perform basic calculations only
- using a remote file access library, only files from the grid can be accessed
- all other kinds of I/O are not necessary for scientific applications and thus prohibited
- indispensable systems calls must be routed to the grid

5.2 Architectural Overview

The SciBy Virtual Machine is an abstract machine executing platform independent bytecodes on a virtual CPU. In many aspects, it is designed similarly to conventional architectures; it includes an Application Binary Interface, an Instruction Set Architecture, and is able to manipulate memory components. The only thing missing in defining the architecture is the hardware. As the VM is supposed to be run on a variety of grid resources, it must be designed to be as portable as possible, thereby supporting many different physical hardware architectures.

Based on the previous sections, the SciBy VM is designed to have 3 fundamental properties: Security, portability, and Performance

5.2.1 Security

Once again, when discussing security, it is important to stress that host system integrity is of primary interest, secondarily data and program isolation from other guest systems; there is no way to protect a guest system from the host system. Host system security is ensured by a virtual machine to isolate untrusted code in a sandbox. Typically, this type of software based fault isolation [WLAG93] focuses on disallowing unsafe instructions access to memory outside the sandbox, illegal instructions, privileged instructions, etc. In SciBy VM, we made the deliberate choice of disallowing system calls, including all types of I/O, altogether. Thus, we only allow instructions that perform transformations on data, control flow instructions, and data movement instructions. Dedicated for scientific applications, there is really no need for system calls, and the only type of I/O necessary, is access to input files and output files; this is achieved using the Remote File Access library, presented in Section 3. Indispensable system calls will be routed back

to the Grid for execution.

A typical sandbox feature, which the SciBy VM also implements, is a Harvard memory model with separate segments for data and code to ensure correct access to data, and preventing jumps to addresses residing in code area. Using this model, the machine is less vulnerable to typical exploits derived from 'illegal' pointer arithmetic to other executable memory segments.

5.2.2 Portability

Portability is obtained by designing a completely platform-independent bytecode and by virtualizing a very broad hardware platform. The instruction set includes all typical instructions for data transformation, control flow, and data movement. To capture most of the physical computer platforms, it is designed as an orthogonal multi-opcode multi-address set of instructions in a 3-operand format, thereby permitting easy translation to 2-operand architectures. Addressing modes include immediate addressing, displacement addressing, register addressing, and indirect addressing.

Virtualization occurs at many levels in various computer subsystems. It typically provides an illusion of hardware configurations that are not physically available, for instance virtual memory which gives each process the illusion of having exclusive access to the entire address space of the machine. With the SciBy VM, much focus is placed on being forward compatible by virtualizing hardware setups of the future. Most notably, as the tendency goes towards more and more registers, the machine provides a virtually unlimited number of registers. Just to provide a number for the compiler, it is set to 16384 128-bit registers. Further, as there is now a shift from 32 bit towards 64 bit architectures, the next step probably being 128 bit, the SciBy VM supports all

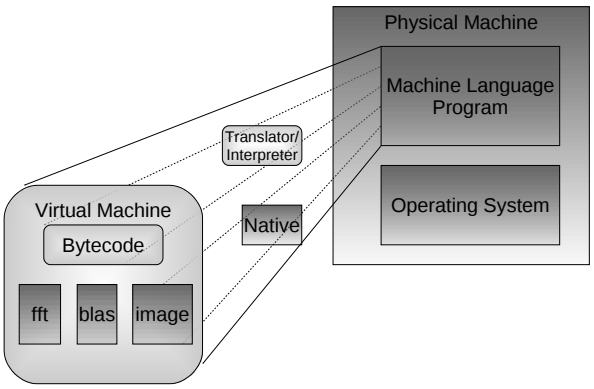


Figure 5.3: An untrusted application converted to bytecode, running in VM with embedded scientific native libraries.

these word sizes. For instance, the SPEs in the Cell BE architecture [CRDI07] already expose 128 bit register lines and a 128 bit instruction set.

5.2.3 Performance

Performance is obtained by augmenting the instruction categories with one essential category: Instructions that can call external native library functions. Executing bytecodes in a virtual machine will incur performance overhead, but the key point here is to utilize the fact that scientific applications spend most of their time in highly optimized core libraries. The characteristics of these applications, for instance bioinformatics, high-energy physics, or image processing analysis, is a small but very time-consuming code size. Typically, these applications set up input and output files, and then enter some dense loop structures in which time-consuming calls to an external library are continuously made. In the SciBy VM, the surrounding code is replaced by the portable bytecode, but with support for calling native optimized libraries in order to achieve near native performance, this is illustrated in Figure 5.3.

Obviously, native libraries are not portable and a version of the machine must be

equipped with statically linked libraries for every architecture, and made available for people willing to participate. However, with the success of a library follows ports to other architectures, and it is a simple and small task to embed a library in the machine.

The bytecode is as such not designed for performance. Therefore, the architectural decisions do not necessarily seek to minimize code density, minimize code size, reduce memory traffic, increase the average number of clock cycles per instruction, or other architectural evaluation measurements, but more for simplicity and portability.

5.2.4 Application Binary Interface

The SciBy VM ABI defines how compiled applications interface with the virtual machine, thus enabling platform independent bytecodes to be executed without modification on the virtual CPU.

At the lowest level, the architecture defines the following machine types arranged in big endian order:

- 8-bit byte
- 16-, 32-, or 64-bit halfword
- 32-, 64-, or 128-bit word
- 64-, 128-, or 256-bit doubleword

In order to support many different architectures, the machine exists in multiple variations with different word sizes. Currently, most desktop computers are either 32- or 64-bit architectures, and it probably won't be long before we see desktop computers with 128-bit architectures. By letting the word size be user-defined, we capture most existing and near-future computers.

Fundamental primitive data types include, all in signed two's complement representation:

- 8-bit character
- integers (1 word)
- single-precision floating point (1 word)
- double-precision floating point (2 words)
- pointer (1 word)

The machine contains a register file of 16384 registers, all 1 word long. This number only serves as a value for having a potentially unlimited amount of registers. The reasons for this are twofold. First of all due to forward compatibility, since the virtual register usage has to be translated to native register usage, in which one cannot tell the upper limit on register numbers. So basically, in a virtual CPU, one should be sure to have more registers than the host system CPU. Currently, 16384 registers should be more than enough, but new architectures tend to have more and more registers. Secondly, for the intended applications, a register-based architecture will outperform a stack-based one[SGBE05]. Generally, registers have proved more successful than other types of internal storage and virtually every architecture designed in the last few decades uses a register architecture.

Register computers exist in 3 classes depending on where ALU instructions can access their operands, register-register architectures, register-memory architectures and memory-memory architectures. The majority of the computers shipped nowadays implement one of those classes in a 2- or 3-operand format. In order to capture as many

computers as possible, the SciBy VM supports all of these variants in a 3-operand format, thereby including 2-operand format architectures in that the destination address is the same as one of the sources.

5.2.5 Instruction Set Architecture

Instructions can broadly be classified as one of four general groups:

- Instructions performing transformation on data: arithmetic, string, floating-point, or logical operations
- Instructions altering program flow: branch, call, return, and loop control instructions
- Instructions performing data movement: load, store, move, push, pop
- System instructions: instructions changing the system's mode

One key element that separates the SciBy VM from conventional machines is the memory model: The machine defines a Harvard [Sto83] memory architecture with separate memory banks for data and instructions. The majority of conventional modern computers use a von Neumann architecture [vN93; Tur36] with a single memory segment for both instructions and data. These machines are generally more vulnerable to the well-known buffer overflow exploits and similar exploits derived from 'illegal' pointer arithmetic to executable memory segments. Furthermore, the machine will support hardware setups that have separate memory pathways, thus enabling simultaneous data and instruction fetches. All instructions are fetched from the instruction memory bank which is inaccessible for applications: All memory accesses from applications are

directed to the data segment. The data memory segment is partitioned in a global memory section, a heap section for dynamically allocated structures, and a stack for storing local variables and function parameters.

Instruction Format

The instruction format is based on *bytecodes* to simplify the instruction stream. The format is as follows: Each instruction starts with a one-byte *operation code* (opcode) followed by possibly more opcodes and ends with zero or more operands, see Figure 5.4. In this sense, the machine is a multi-opcode multi-address machine. Having only a single one-byte opcode limits the instruction set to only 256 different instructions, whereas multiple opcodes allows for nested instructions, thus increasing the number of instructions exponentially. A multi-address design is chosen to support more types of hardware.

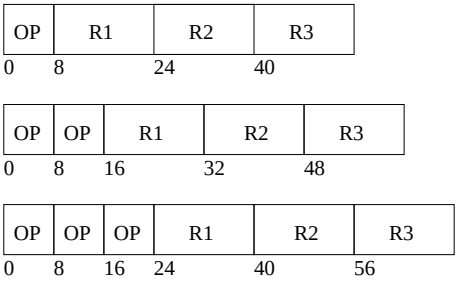


Figure 5.4: Examples of various instruction formats on register operands.

Addressing Modes

Based on the popularity of addressing modes found in recent computers [HP02], we have selected 4 addressing modes for the SciBy VM, all listed below.

- Immediate addressing: The operand is an immediate, for instance MOV R1 4 which moves the number 4 to register 1.
- Displacement addressing: The operand is an offset and a register pointing to a base address, for instance ADD R1 R1 4(R2) which adds to R1 the value found 4 words from the address pointed out by R2.
- Register addressing: Operand is a register, for instance MOV R1 R2
- Register indirect addressing: Address part is a register containing the address of an operand, for instance ADD R1, R1, (R2), which adds to R1 the value found at the address pointed out by R2.

Instruction Types

Since the machine defines a Harvard architecture, it is important to note that data movement is carried out by `LOAD` and `STORE` operations which operate on words in the data memory bank. `PUSH` and `POP` operations are available for accessing the stack.

Table 5.1 summarizes the most basic instructions available in the SciBy VM. Almost all operations are simple 3-address operations with operands, and they are chosen to be simple enough to be directly matched by native hardware operations.

Instruction group	Mnemonic
Moves	load, store
Stack	push, pop
Arithmetic	add, sub, mul, div, mod
Boolean	and, or, xor, not
Bitwise	and, or, shl, shr, ror, rol
Compare	tst, cmp
Control	halt, nop, jmp, jsr, ret, br, be_eq, br_lt, etc

Table 5.1: Basic Instruction Set of the SciBy VM

While these instructions are found in virtually every computer, they exist in many different variations using various addressing modes for each operand. To accommodate this and assist the compiler as much as possible, the SciBy VM provides regularity by making the instruction set orthogonal on both operations, data types, and the addressing modes. For instance the 'add' operation exists in all 16 combinations of the 4 addressing modes on the two source registers for both integers and floating points. Thus, the encoding of an 'add' instruction on two immediate source operands takes up 1 byte for choosing arithmetic, 1 byte to select the 'add' on two immediates, 2 bytes to address one of the 16384 registers as destination register and then 16 bytes for each of the immediates, yielding a total instruction length of 36 bytes.

5.2.6 Libraries

In addition to the basic instruction set, the machine implements a number of basic libraries for standard operations like floating-point arithmetic and string manipulation. These are extensions to the virtual machine and are provided on an per-architecture basis as statically linked native libraries optimized for specific hardware.

As explained above, virtual machines introduce a performance overhead in the translation phase from virtual machine object code to the native hardware instructions of the underlying real machine. The all-important observation here is that scientific applications spend most of their running time executing 'scientific instructions' such as string operations, linear algebra, fast fourier transformations, or other library functions. Hence, by providing optimized native libraries, we can take advantage of the synergy between algorithms, the compiler translating them, and the hardware executing them.

Equipping the machine with native libraries for the most prevalent scientific algo-

rithms and enabling future support for new libraries increases the number of potential instructions drastically. To address this problem, multiple opcodes allows for nested instructions as shown in Figure 5.5. The basic instructions are accessible using only one opcode, whereas a floating point operation is accessed using two opcodes, i.e. *FP_lib FP_sub R1 R2 R3*, and finally, if one wishes to use the `WFTA` instruction from the `FFT_2` library, 3 opcodes are necessary: *FFT_lib FFT_2 WFTA args*.

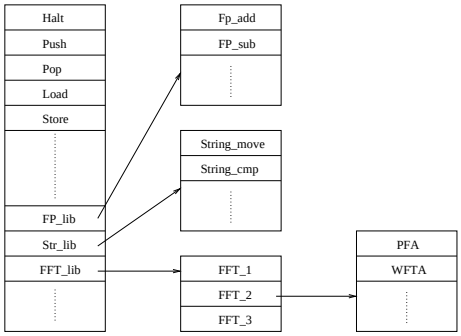


Figure 5.5: Native libraries as extension to the instruction set

As no system or I/O calls are available, the machine uses the MiG-RFA 3 library to access files directly on the MiG file servers on an on-demand basis. Using this strategy, an application can start immediately, and only the needed fragments of the files it accesses are transferred.

5.2.7 Final Notes

Ending the discussion of the architecture, it is important to re-emphasize that all focus in the design of the bytecode is on portability and security. Regarding security, the Harvard design is chosen to avoid internal exploits that possibly could lead to a malicious program breaking out of the sandbox, and by excluding all I/O and system calls, there is no way for programs in the sandbox to communicate with the host system.

Designing an architecture includes a lot of trade-offs, and even though many of these issues are zeroed by the interpreter or translator, the proposed bytecode is far from optimal by normal architecture metrics. For instance, when evaluating the architecture, one might find that:

- Having a 3-operand instruction format may give unnecessarily large code size in some circumstances
- Studies may show that the displacement addressing mode is typically used to nearby addresses, thereby suggesting that these instructions only need a few bits for the operand
- Using register-register instructions may give unnecessarily high instruction count in some circumstances
- Using bytecodes increases the code density
- Variable instruction encoding decreases performance

However, the key point is that we target only a special type of applications on a very broad hardware platform. Performance-wise, the bytecode itself has no significance.

5.3 Implementation

In order to validate the proposed ideas and justify further development, an assembler and a very simple bytecode interpreter were implemented. The assembler translates compiled code in assembly format into the machine's native bytecode, which the interpreter executes. In the code section below, the bytecode is read from the file descriptor, *fd*. The interpreter simply reads a byte from the code area, consults a call-table to find

out which type of instruction the byte corresponds to, for instance `arithmetic`, and enters the call-table of the instruction type. The central code the interpreter is thus very simple (return code check and memory allocations have been left out for readability):

```
unsigned char instruction, *ip;
function *call_table;
call_table[0x00] = control_ops;
...
call_table[0x06] = arithmetic_ops;
...
text = mmap (0, statbuf.st_size, PROT_READ, MAP_SHARED, fd, 0);
ip = text //instruction pointer
while (1){
    instruction = get_1_byte(&ip);
    call_table[instruction]((void **)&ip);
}
```

Once in the arithmetic section, the interpretation proceeds by reading the next byte, looks it up in a similar call-table populated with the arithmetic instructions. For instance, if the bytecode translates to the `add` variant performed on three registers, more bytes will be read until all registers are determined, and finally, the instruction can be performed:

```
arithmetic_instructions[0x00] = addi_r_r;
...
void arithmetic_ops(void **p){
  unsigned char *pc = *p;
  unsigned char sub_instruction;
  sub_instruction = get_1_byte(&pc);
```

```
  arithmetic_instructions[sub_instruction]((void **)&pc);
  *p = pc;
}
static void addi_r_r(void **p){
  unsigned char *pc = *p;
  int dest = get_2_bytes(&pc);
  int src1 = get_2_bytes(&pc);
  int src2 = get_2_bytes(&pc);
  registers[dest] = registers[src1] + registers[src2];
  *p = pc;
}
```

In addition to all the standard instructions, the call-table is also populated with an entry for every supported native library:

```
call_table[0x07] = fftw_sciby_ops;
call_table[0x08] = cuda_sciby_ops;
call_table[0x08] = gsl_matrix_sciby_ops;
call_table[0x0a] = fourier_sciby_ops;
call_table[0x0b] = dip_sciby_ops;
...
```

The instruction tables of each of these are populated with an instruction for every function available in the API of the library. For instance, to apply an *Adaptive Smoothing Filter* on an image using the `fourier` [Cel08] image library, the following instructions from the API are necessary to read in the image, apply the filter, write it back, and finally free the image:

```
fourier_sciby_instructions[0x00] = fourier_sciby_read_img;
fourier_sciby_instructions[0x01] = fourier_sciby_filter_adap_smooth;
```

```
fourier_sciby_instructions[0x02] = fourier_sciby_write_img;
fourier_sciby_instructions[0x03] = fourier_sciby_free_img;
...
void fourier_sciby_ops(void **p){
  unsigned char *pc = *p;
  unsigned char sub_instruction;
  sub_instruction = get_1_byte(&pc);
  fourier_sciby_instructions[sub_instruction]((void **)&pc);
  *p = pc;
}
```

Unlike the standard instruction functions, the corresponding functions for library instructions perform the actual work by calling the native library function, thus off-loading computational intensive instructions to native code.

5.4 Related Work

GridBox [DSG04] aims at providing a secure execution environment for grid applications by means of a sandbox environment and Access Control Lists. The execution environment is restricted by the *chroot* command which isolates each application in a separate file system space. In this space, all system calls are intercepted and checked against pre-defined Access Control Lists which specify a set of allowed and disallowed actions. In order to intercept all system calls transparently, the system is implemented as a shared library that gets preloaded into memory before the application executes. The drawback of the GridBox library is the requirement of a UNIX host system and application and it does not work with statically linked applications. Further, this kind of isolation can be opened if an intruder gains system privileges leaving the host system unprotected.

Secure Virtual Grid (SVGrid) [ZBP05] isolates each grid applications in its own instance of a Xen virtual machine whose file system and network access requests are forced to go through the privileged virtual machine monitor where the restrictions are checked. Since each grid virtual machine is securely isolated from the virtual machine monitor from which it is controlled, many levels of security has to be opened in order to compromise the host system, and the system has proved its effectiveness against several malicious software tests. The performance of the system is also above acceptable with a very low overhead. The only drawback is that while the model can be applied to other operating systems than Linux, it still makes use of platform-dependent virtualization software.

Java and the Microsoft Common Language Infrastructure are similar solutions trying to enable applications written in the Java programming language or the Microsoft .Net framework, respectively, to be used on different computer architectures without being rewritten. They both introduce an intermediate platform independent code format (Java bytecode and the Common Intermediate Language respectively) executable by hardware-specific execution environments (the Java Virtual Machine and the Virtual Execution System respectively). While these solution have proved suitable for many application domains, performance problems and their requirement of a specific programming language class rarely used for scientific applications disqualifies the use of these virtual machines for this project.

NestedVM [AM04] provides binary translation from unsafe machine code into safe java bytecodes. Compilation from source code is left to standard compilers such as GCC, which are used to compile from any source language into MIPS machine language. The MIPS instructions are then mapped by the NestedVM into Java bytecodes and executed. According to their own benchmarks, the system incurs a slow-down of a factor 3 compared to native execution.

5.5 Experiments & Results

All sample programs are hand-written in assembly code and processed by the assembler that translates them into the bytecodes executable by the interpreter. The first test is a typical example of the scientific applications the machine targets: A Fast Fourier Transform (FFT).

5.5.1 Fast Fourier Transform

FFT is an obvious choice for evaluating the SciBy VM, since it is a fundamental kernel in so many scientific applications, for instance data compression, fluid dynamics, seismic imaging, image processing, computer tomography, data filtering, spectral analysis, and digital signal processing. The core of these applications is the necessity of computing Fourier transforms, and the performance of this type of application relies heavily on the routines available for performing the transforms.

The program first computes 10 transforms on a vector of varying sizes, then checksums the transformed vector to verify the result. In order to test the performance of the virtual machine, the program is also implemented in C to get the native base line performance, and in Java to compare the results of the SciBy VM with an existing widely used virtual machine.

The experiments are carried out on the following machines:

- A 1.86 GHz Intel Pentium M, 2 MB cache, 512 MB RAM
- A dual core 2.2 GHz AMD Athlon 4200 64-bit, 512 kB cache per core, 4 GB RAM
- A dual quad core Intel Xeon, 1.60 GHz, 4 MB cache per core, 8 GB RAM

The C and SciBy VM programs make use of the `fftw` library[FJ05], while the Java version uses an FFT algorithm from the SciMark suite[PM]. Obviously, this test is highly unfair in disfavor of the Java version for several reasons. Firstly, the fftw library is well-known to give the best performance, and comparing hand-coded assembly with compiler-generated high-level

Vector size	Native	SciBy VM	Java
524288	1.535	1.483	7.444
1048576	3.284	3.273	19.174
2097152	6.561	6.656	41.757
4194304	14.249	14.398	93.960
8388608	29.209	29.309	204.589

Table 5.2: Comparison of the performance of an FFT application on a 1.86 GHz Intel Pentium M processor, 2MB cache, 512 MB RAM

Vector size	Native	SciBy VM	Java
524288	0.879	0.874	4.867
1048576	1.857	1.884	10.739
2097152	3.307	3.253	23.520
4194304	6.318	6.354	50.751
8388608	13.045	12.837	110.323

Table 5.3: Comparison of the performance of an FFT application on a dual core 2.2 GHz AMD Athlon 4200 64-bit, 512 kB cache per core, 4GB RAM

language performance is a common pitfall. However, even though Java-wrappers for the fftw library exist, it is essential to put these comparisons in a grid context. If the grid resources were to run the scientific applications in a Java Virtual Machine, the programmers - the grid users - would not be able to take advantage of the native libraries, since allowing external library calls breaks the security of the JVM. Thereby, the isolation level between the executing grid job and the host system is lost[†]. In the proposed virtual machine, these libraries are an integrated part of the machine, and using them is perfectly safe.

As shown in Table 5.2 the FFT application is run using different vector size, $2^{19}, ..., 2^{23}$. The results in all experiments are the average of 3 consecutive runs, all measured in seconds using the Linux `time` command. The results show that the SciBy VM is on-par with native execution, and that the Java version is clearly outperformed.

Since the fftw library is multithreaded, we repeat the experiment on a dual core machine and on a quad dual-core machine. The results are shown in Table 5.3 and Table 5.4.

[†]In fact there is a US Patent (#6862683) on a method to protect native libraries

Vector size	Native	SciBy VM	Java
524288	0.650	0.640	4.955
1048576	1.106	1.118	12.099
2097152	1.917	1.944	27.878
4194304	3.989	3.963	61.423
8388608	7.796	7.799	134.399

Table 5.4: Comparison of the performance of an FFT application on a quad dual-core Intel Xeon CPU, 1.60 GHz, 4MB cache per core, 8GB RAM

From these results it is clear that for this application there is no overhead in running it in the virtual machine. It has immediate support for multi-threaded libraries, and therefore the single-threaded Java version is even further outperformed on multi-core architectures.

5.5.2 Image Processing

Image processing is widely used in many scientific applications, such as medical imaging, computer graphics rendering, sensing and detection systems, and in general, over the last few years it is becoming a topic of interest for a broad scientific community. Using `fourier` [Cel08], a portable image processing and analysis library, we write a sample application that applies a series of transformations, for instance `gaussian adaptive smoothing` filter, on a raw 2592x1944 pixel `pgm` image. We then run the application in 4 different setups:

- Native: Run natively, with the image in the local file system
- SciBy VM: Run inside the SciBy VM with the image in the local file system (suspending the local file access restriction)
- Native + RFA: Run natively, using the Remote File Access library to access the file 200 km away (standard 5 Mbps ADSL-line).
- SciBy VM + RFA: Run inside the SciBy VM using the RFA library as above.

These tests are performed on 4 architectures:

Table 5.5: Results from the fourier application on a single-core host machine including transfer time of the image data from Grid

Arch.	Native	SciBy	Native+RFA	SciBy+RFA
Core 2	96.36	96.41	96.87	96.90
AMD	78.93	78.89	78.75	79.01
PPC	166.18	164.25	167.76	166.21
Mac	58.72	59.05	59.55	60.01

- An Intel Core 2 1.86 GHz processor, 2GB memory, running 32 bit Ubuntu linux,
- An AMD Athlon 64-bit processor 3000+, 1GB memory, running Debian-amd64
- A 3.2 GHz PowerPC 64-bit processor, 2 hardware threads, from the Cell Broadband Engine, running 32-bit Yellow Dog Linux[‡]
- an Intel Core 2 2.4 GHz processor, 4GB memory, running Mac OSX.

Table 5.5 shows that there is no overhead when running inside the virtual machine in any of the setups. Since the entire file is used in an unbalanced fashion, there is no gain from using the RFA library. The time to transfer the image using `curl` and `lighttpd` was 46.293 seconds, which is exactly the difference between the executions with and without RFA. Thus, a staging technique would be equal to accessing the file remotely.

Next, to illustrate utilization of a multi-core architecture, we use the `diplib` [vG00] image processing library, which is multi-threaded. In this test, we apply the very compute-intensive second order derivative Laplace filter on the image, and execute on the 8-core Intel Xeon 1.60 GHz with 8 GB memory. Using the `taskset` command, the experiment is carried out using 1,2,4, and 8 cores.

From the results in Table 5.6, we can once again conclude that there is no overhead from using the virtual machine. And, there is immediate support for multi-core utilization. Again,

[‡]The `fourier` image library does not utilize the SPEs in the Cell BE.

Table 5.6: Results from the `diplib` application on an 8-core host machine. The image transfer time, 46 seconds, is not included

Cores	Native	SciBy	Native+RFA	SciBy+RFA
1	101.53	101.34	147.47	147.80
2	51.36	51.94	97.76	97.60
4	27.74	27.63	73.23	73.54
8	15.14	15.23	61.87	61.90

Table 5.7: Results from the video processing, including transfer time of the image from the Grid

Arch.	Native	SciBy	Native+RFA	SciBy+RFA
Core 2	376.58	377.11	261.81	262.01
AMD	260.24	260.13	192.89	193.11

since the `diplib` image reads the entire image before it processes it, there is no gain from the remote file access library.

To wrap up image processing performance tests, a final example uses the `ffmpeg` [Bel00] library to decode frames from a video file. Each decoded frame is then transformed using the `Imlib2` [Hai99] image library. In this case, we just a apply a simple blurring effect to every frame. Since the `ffmpeg` balances I/O and processing, i.e. it consecutively reads data for a single frame, and then decodes it, the RFA library can take advantage of the prefetching, thus having every frame available in the instant it is needed. Therefore, as shown in 5.7, using the RFA library is faster than using local file access. The test is performed on the Intel and AMD computers only.

5.5.3 Basic Linear Algebra Subroutines

`BLAS` (Basic Linear Algebra Subroutines)is widely used for high-performance computing and benchmarking. `BLAS` is a set of efficient routines for most of the basic vector and matrix operations. They are widely used as the basis for other high quality linear algebra software packages,

Table 5.8: Results from the BLAS benchmark

Arch.	Native	SciBy VM
Core 2	38.466	38.211
AMD	46.340	46.870
PPC	43.513	43.662
Mac	36.344	36.492

for instance `lapack` and `linpack`. In this test, we perform a series of operations from the ATLAS [WP05] and GotoBLAS [GvdG02] libraries on a 500 by 500 matrix. It is all computed in memory, so there is no file access. The results displayed in 5.8 once again show that the SciBy VM achieves near native speed.

5.6 Conclusion

Bytecodes and a virtual machine executing them provides platform-independence at the cost of performance. The SciBy VM introduces a hybrid model, where the machine executes mobile bytecodes and uses native optimized libraries to mitigate the performance drawback. Thus, the model applies best to applications that spend most of their time in the libraries, which is the case for scientific applications.

Using typical scientific libraries, the performance of the machine has been evaluated and found to be on par with native execution. For other types of applications, the machine will currently perform poorly.

Chapter 6

Conclusions

During the last decade, the two main technologies on which this project is founded, Virtual Machines and Grid Computing, have both had their peak on the hype curve. First, Grid Computing emerged as the solution to most computationally intensive scientific applications. Next, VMs reemerged with many properties such as server consolidation, better multi-core utilization, application mobility, and sandbox abilities.

As the technologies became more widely demonstrated and accepted, it also became apparent that there were some drawbacks to the available products. In terms of performance, Grid Computing was outpaced by Volunteer Computing systems, and while the specification of a Grid is wide enough to include standard desktop computers, none of the available Grid systems have adopted this resource platform. Merging Volunteer Computing and Grid Computing systems is a big step towards realizing one of the initial ideas of Grid Computing, namely harnessing idle CPU power from all types of computer resources, and putting them on tap for world-wide sharing.

Virtual machines can bridge the architectural boundaries between different resource types, but the drawback of virtual machines is either performance or portability. While virtual machines can exhibit reasonable performance when the guest and host systems are compatible, the

portability is limited to a single architecture. However, to harness the compute power from an ever-increasing farm of architectures and use it for scientific research, the application mobility and sandbox properties of virtual machines make them ideal for deployment in a combined Grid and Volunteer Computing system.

The thesis has presented two virtual machines, the MiG-SSS and the SciBy VM. The MiG-SSS is based on existing virtual machines and aimed at Windows and Linux desktop PCs, which were the most popular desktop devices at the time when it was launched in 2006. Disguised as a screen saver, the machine harnesses the idle time CPU cycles from privately owned Internet-connected PCs. In the latest edition, the MiG-SSS is also available as a Windows Service, where it runs constantly as a low-priority process in the background.

The SciBy VM is a virtual machine designed specifically for scientific applications. By taking advantage of the fact that scientific applications spend most of the execution time doing linear algebra, fast fourier transforms, string operations, and other library functions, this solution introduces a hybrid machine that executes platform independent bytecodes and has the ability of calling native libraries. And, by equipping the machine with a remote file access library, the need for host system interaction has been completely eliminated. The portability of the machine is only limited by the availability of scientific libraries, and experiments have shown that the machine's performance is on par with native execution.

6.1 Future Work

Throughout the course of this project, many sub-projects have been spawned, and many refinements and new ideas are ready in the pipeline. Regarding the MiG-SSS, the integration of a suspend-migrate-resume feature in the MiG system is the first task. Secondly, the dynamic configuration and loading of runtime environments needs to be implemented. Finally, improvements on the screen saver graphics and job count monitoring have been requested from the user

community.

The SciBy VM still needs hardening, and several future directions exist before the model can be deployed:

Compiler Most notably, a compiler is necessary to translate the source code from a high level programming language into the assembly code. `GCC` is the most widely used compiler suite, and by providing a detailed specification of the machine ABI and a general machine description, it is possible to port `GCC` to new architectures [Nil00]. Another possibility is to use `lcc` which is designed to be retargetable. In addition to a compiler, a profiler and a debugger will also be convenient tools for developers.

GPU Support For some types of scientific applications, Graphical Processing Units are much more powerful than CPUs [OHL+08]. Initial experiments have shown how to integrate the CUDA [NBGS08] computing architecture in the machine, yet more work is needed to fully support GPUs.

Libraries Only a handful scientific libraries have been integrated in the machine. Support for many more libraries is necessary for the machine to be successful. Moreover, libraries for distributed shared memory systems are frequently used in scientific applications and should be supported by the machine as well.

Translator In the event that faster execution of the bytecode becomes essential, another approach than the interpreter would be to use just-in-time compilation techniques [Ayc03; Kra98] that generally perform better than bytecode interpreters. However, as scientific applications typically execute for a long time, the load time overhead is likely to be amortized by the total wall-clock time.

Bibliography

[AA06] Keith Adams and Ole Agesen, *A comparison of software and hardware techniques for x86 virtualization*, ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems (New York, NY, USA), ACM, 2006, pp. 2–13.

[ABB+01] Bill Allcock, Joe Bester, John Bresnahan, Ann L. Chervenak, Ian Foster, Carl Kesselman, Sam Meder, Veronika Nefedova, Darcy Quesnel, and Steven Tuecke, *Data management and transfer in high-performance computational grid environments*, Parallel Computing Journal **28** (2001), 749–771.

[ACK+02] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer, *Seti@home: an experiment in public-resource computing*, Commun. ACM **45** (2002), no. 11, 56–61.

[AGG02] Ken Arnold, Guang R. Gao, and Sudipto Ghosh, *Java/jini technologies and high-performance pervasive computing*, SPIE- International Society for Optical Engineering, 2002.

[AM04] Brian Alliet and Adam Megacz, *Complete translation of unsafe native code to safe bytecode*, IVME '04: Proceedings of the 2004 workshop on Interpreters, virtual machines and emulators (New York, NY, USA), ACM, 2004, pp. 32–41.

[AMD05] AMD, *Amd64 virtualization codenamed "pacifica" technology, secure virtual machine architecture reference manual*, 2005.

[And03] David P. Anderson, *Public computing: Reconnecting people to science*, Proceedings of Conference on Shared Knowledge and the Web, 2003, pp. 17–19.

[And04] ———, *Boinc: A system for public-resource computing and storage*, GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (Washington, DC, USA), IEEE Computer Society, 2004, pp. 4–10.

[AV05] Rasmus Andersen and Brian Vinter, *Transparent remote file access in the minimum intrusion grid*, WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (Washington, DC, USA), IEEE Computer Society, 2005, pp. 311–318.

[AV06] ———, *Harvesting idle windows cpu cycles for grid computing*, GCA (Hamid R. Arabnia, ed.), CSREA Press, 2006, pp. 121–126.

[AV07] ———, *Direct application access to grid storage: Research articles*, Concurr. Comput. : Pract. Exper. **19** (2007), no. 9, 1287–1298.

[AV08] ———, *The scientific byte code virtual machine*, GCA (Hamid R. Arabnia, ed.), CSREA Press, 2008, pp. 175–181.

[Ayc03] John Aycock, *A brief history of just-in-time*, ACM Comput. Surv. **35** (2003), no. 2, 97–113.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield, *Xen and the art of virtualization*, SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles (New York, NY, USA), ACM Press, 2003, pp. 164–177.

[Bel00] Fabrice Bellard, *ffmpeg*, 2000, `http://www.ffmpeg.org/`.

[Bel05] Fabrice Bellard, *Qemu, a fast and portable dynamic translator*, 2005, pp. 41–46.

[CCWY05] Brad Calder, Andrew A. Chien, Ju Wang, and Don Yang, *The entropia virtual machine for desktop grids*, VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments (New York, NY, USA), ACM, 2005, pp. 186–196.

[Cel08] M. Emre Celebi, *fourier*, 2008, `http://mac.softpedia.com/get/Development/Libraries/Fourier.shtml`.

[CHPM07] Neil Cafferkey, Philip D. Healy, David A. Power, and John P. Morrison, *Job management in webcom*, ISPDC '07: Proceedings of the Sixth International Symposium on Parallel and Distributed Computing (Washington, DC, USA), IEEE Computer Society, 2007, p. 6.

[CIS07] CIS, *Virtual machine security guidelines*, 2007.

[CRDI07] T. Chen, R. Raghavan, J. N. Dale, and E. Iwata, *Cell broadband engine architecture and its first implementation: a performance view*, IBM J. Res. Dev. **51** (2007), no. 5, 559–572.

[dANV+05] Marcos Dias de Assuno, Krishna Nadiminti, Srikumar Venugopal, Tianchi Ma, and Rajkumar Buyya, *An integration of global and enterprise grid computing: Gridbus broker and xgrid perspective*, In Proceedings of the 4th International Conference on Grid and Cooperative Computing (GCC 2005), LNCS, Springer-Verlag, 2005.

[DE99] Sophia Drossopoulou and Susan Eisenbach, *Describing the semantics of java and*

proving type soundness, Formal Syntax and Semantics of Java (London, UK), Springer-Verlag, 1999, pp. 41–82.

[Den71] Peter J. Denning, *On modeling program behavior*, AFIPS '71 (Fall): Proceedings of the November 16-18, 1971, fall joint computer conference (New York, NY, USA), ACM, 1971, pp. 937–944.

[DSG04] Evgueni Dodonov, Joelle Quaini Sousa, and Hélio Crestana Guardia, *Gridbox: securing hosts from malicious and greedy applications*, MGC '04: Proceedings of the 2nd workshop on Middleware for grid computing (New York, NY, USA), ACM Press, 2004, pp. 17–22.

[EEH+03] P. Eerola, M. Ellert, J. R. Hansen, A. Konstantinov, B. Knya, J. L. Nielsen, O. Smirnova, and A. Wnnen, *The nordugrid: Building a production grid in scandinavia*, 2003.

[Far06] Gilles Farrache, *bbftp, large files transfer protocol*, 2006.

[FC08] Bryan Ford and Russ Cox, *Vx32: lightweight user-level sandboxing on the x86*, ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference (Berkeley, CA, USA), USENIX Association, 2008, pp. 293–306.

[FDF03] Renato J. Figueiredo, Peter A. Dinda, and Fortes, *A case for grid computing on virtual machines*, ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems (Washington, DC, USA), IEEE Computer Society, 2003.

[Fer06] Peter Ferrie, *Attacks on virtual machine emulators*, 2006.

[FJ05] Matteo Frigo and Steven G. Johnson, *The design and implementation of FFTW3*,

Proceedings of the IEEE **93** (2005), no. 2, 216–231, special issue on "Program Generation, Optimization, and Platform Adaptation".

[Fos05] Ian T. Foster, *Globus toolkit version 4: Software for service-oriented systems.*, NPC (Hai Jin, Daniel A. Reed, and Wenbin Jiang, eds.), Lecture Notes in Computer Science, vol. 3779, Springer, 2005, pp. 2–13.

[GBD+94] A. Geist, A. Beguelin, Jack Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *Pvm parallel virtual machine, A user's guide and tutorial for networked parallel computing*, MIT Press, Cambridge, Mass., 1994.

[GFB+04] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall, *Open MPI: Goals, concept, and design of a next generation MPI implementation*, Proceedings, 11th European PVM/MPI Users' Group Meeting (Budapest, Hungary), September 2004, pp. 97–104.

[Gol73] R. P. Goldberg, *Architecture of virtual machines*, Proceedings of the workshop on virtual computer systems (New York, NY, USA), ACM, 1973, pp. 74–112.

[GS08] William L. George and Jacob Scott, *Screen saver science: Realizing distributed parallel computing with jini and javaspaces*.

[Gu06] Yunhong Gu, *Udt: Udp-based data transfer protocol*, 2006.

[GvdG02] Kazushige Goto and Robert van de Geijn, *On reducing TLB misses in matrix multiplication*, Tech. Report TR-2002-55, University of Texas, November 2002, FLAME working note #9, `http://www.tacc.utexas.edu/resources/software`.

[Hai99] Carsten Haitzler, *Imlib2*, 1999, `http://docs.enlightenment.org/api/imlib2/html/`.

[HKM+88] John H. Howard, Michael L. Kazar, Sherri G. Menees, A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West, *Scale and performance in a distributed file system*, ACM Transactions on Computer Systems **6** (1988), 51–81.

[HP02] John L. Hennessy and David A. Patterson, *Computer architecture: A quantitative approach*, Morgan Kaufmann, May 2002.

[JPMC00] Keith Power John P. Morrison and Neil Cafferkey, *Cyclone: A cycle brokering system to harvest wasted processor cycles*, Parallel and Distributed Computing Techniques and Applications, 2000.

[Ker99] A. Keromytis, *The keynote trust-management system, version 2*, IETF RFC **2704** (1999), 164–173.

[KF98] Carl Kesselman and Ian Foster, *The grid: Blueprint for a new computing infrastructure*, Morgan Kaufmann Publishers, November 1998.

[Kra98] A. Krall, *Efficient javavm just-in-time compilation*, PACT '98: Proceedings of the 1998 International Conference on Parallel Architectures and Compilation Techniques (Washington, DC, USA), IEEE Computer Society, 1998, p. 205.

[KV05] Henrik Hoey Karlsen and Brian Vinter, *Minimum intrusion grid - the simple model*, WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (Washington, DC, USA), IEEE Computer Society, 2005, pp. 305–310.

[KV06] Henrik Hoey Karlsen and Brian Vinter, *Vgrids as an implementation of virtual organizations in grid computing*, WETICE '06: Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (Washington, DC, USA), IEEE Computer Society, 2006, pp. 175–180.

[LLM88] Michael Litzkow, Miron Livny, and Matthew Mutka, *Condor - a hunter of idle workstations*.

[LSSP09] Stefan M. Larson, Christopher D. Snow, Michael Shirts, and Vijay S. Pande, *Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology*, 2009.

[LY99] Tim Lindholm and Frank Yellin, *The java(tm) virtual machine specification (2nd edition)*, Prentice Hall PTR, April 1999.

[MO98] H. Mcghan and M. O'Connor, *Picojava: a direct execution engine for java bytecode*, Computer **31** (1998), no. 10, 22–30.

[MOH03] John P. Morrison, Padraig J. O'Dowd, and Philip D. Healy, *Searching rc5 keyspaces with distributed reconfigurable hardware*, Engineering of Reconfigurable Systems and Algorithms (Toomas P. Plaks, ed.), CSREA Press, 2003, pp. 269–272.

[MP05] Thomas Mailund and Christian N. S. Pedersen, *Initial experiences with generecon on mig*, In Proceedings of the 2005 International Conference on Grid Computing and Applications (GCA05), Monte Carlo Resort, Las Vegas, 2005.

[MPK03] John P. Morrison, David A. Power, and James J. Kennedy, *An evolution of the webcom metacomputer*, J. Math. Model. Algorithms **2** (2003), no. 3, 263–276.

[MSS+97] Albert Alexandrov Maximilian, Ian E. Schauser, Chris J. Scheima, Albert D. Alex, Albert D. Alex, Maximilian Ibel, Maximilian Ibel, Klaus E. Schauser, and Chris J. Scheiman, *Extending the operating system at the user level: the ufo global file system*, 1997.

[MWG] Erik Meijer, Redmond Wa, and John Gough, *Microsoft clr overview*.

[MY01] Sathiamoorthy Manoharan and Chaitanya Reddy Yavasani, *Experiments with sequential prefetching*, HPCN Europe 2001: Proceedings of the 9th International Conference on High-Performance Computing and Networking (London, UK), Springer-Verlag, 2001, pp. 322–331.

[NBGS08] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron, *Scalable parallel programming with cuda*, Queue **6** (2008), no. 2, 40–53.

[Nil00] Hans-Peter Nilsson, *Porting gcc for dunces*, 2000.

[NS03] Nicholas Nethercote and Julian Seward, *Valgrind: A program supervision framework*, Electronic Notes in Theoretical Computer Science **89** (2003), no. 2.

[NS07] Nicholas Nethercote and Julian Seward, *Valgrind: a framework for heavyweight dynamic binary instrumentation*, SIGPLAN Not. **42** (2007), no. 6, 89–100.

[OHL+08] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, *Gpu computing*, Proceedings of the IEEE **96** (2008), no. 5, 879–899.

[Par06] Parallels, *Parallels desktop 4.0 for mac*, 2006.

[PBW+05] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten, *Scalable molecular dynamics with namd.*, J Comput Chem **26** (2005), no. 16, 1781–1802.

[PC06] Virtual PC, *Microsoft virtual pc*, 2006.

[PG74] Gerald J. Popek and Robert P. Goldberg, *Formal requirements for virtualizable third generation architectures*, Commun. ACM **17** (1974), no. 7, 412–421.

[PM] Roldan Pozo and Bruce Miller, *Scimark 2.0*, `http://math.nist.gov/scimark2/`.

[PSB+00] B. Pawlowski, S. Shepler, C. Beame, B. Callaghan, M. Eisler, D. Noveck, D. Robinson, and R. Thurlow, *The nfs version 4 protocol*, 2000.

[RI00] John Scott Robin and Cynthia E. Irvine, *Analysis of the intel pentium's ability to support a secure virtual machine monitor*, SSYM'00: Proceedings of the 9th conference on USENIX Security Symposium (Berkeley, CA, USA), USENIX Association, 2000, pp. 10–10.

[RV07] Martin Rehr and Brian Vinter, *The one-click grid-resource model.*, HPCC (Ronald H. Perrott, Barbara M. Chapman, Jaspal Subhlok, Rodrigo Fernandes de Mello, and Laurence Tianruo Yang, eds.), Lecture Notes in Computer Science, vol. 4782, Springer, 2007, pp. 296–308.

[RV08a] ———, *Application porting and tuning on the cell-be processor*, Proceedings of PARA '08, May 2008, Extended abstract.

[RV08b] ———, *The ps3 grid-resource model.*, GCA (Hamid R. Arabnia, ed.), CSREA Press, 2008, pp. 90–95.

[Sar98] Luis F. G. Sarmenta, *Bayanihan: Web-based volunteer computing using java*, In Second International Conference on World-Wide Computing and its Applications, 1998, pp. 444–461.

[Sch01] Pascal Schmidt, *ttylinux*, 2001.

[SGBE05] Yunhe Shi, David Gregg, Andrew Beatty, and M. Anton Ertl, *Virtual machine showdown: stack versus registers*, VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments (New York, NY, USA), ACM, 2005, pp. 153–163.

[Smi82] Alan Jay Smith, *Cache memories*, ACM Comput. Surv. **14** (1982), no. 3, 473–530.

[Ste] Daniel Stenberg, *curl*, `http://curl.haxx.se`.

[Sto83] E. L. Stoll, *Mark i*, Ralston, Anthony; Reilly, Edwin D., Encyclopedia of computer science and engineering (2nd ed.), New York: Van Nostrand Reinhold Company Inc (1983), 916–917.

[Sym99] Don Syme, *Proving java type soundness*, Formal Syntax and Semantics of Java (London, UK), Springer-Verlag, 1999, pp. 83–118.

[Tur36] A. M. Turing, *On computable numbers, with an application to the entscheidungsproblem*, Proc. London Math. Soc. **2** (1936), no. 42, 230–265.

[UNR+05] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. M. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, and L. Smith, *Intel virtualization technology*, Computer **38** (2005), no. 5, 48–56.

[VABK06] Brian Vinter, Rasmus Andersen, Jonas Bardino, and Henrik H Karlsen, *Massive cycle harvesting for computational chemistry*, 2006.

[vG00] Michael van Ginkel, *diplib*, 2000, `http://www.diplib.org`.

[Vin05] Brian Vinter, *The Architecture of the Minimum intrusion Grid (MiG)*, Communicating Process Architectures 2005, sep 2005, pp. –.

[Vin07] Brian Vinter, *The grid taken literally*, 2007.

[VL97] Steven P. VanderWiel and David J. Lilja, *When caches aren't enough: Data prefetching techniques*, Computer **30** (1997), no. 7, 23–30.

[VMW99] VMWare, *Vmware workstation*, 1999.

[VMW06] ———, *Vmware player*, 2006.

[vN93] John von Neumann, *First draft of a report on the edvac*, IEEE Ann. Hist. Comput. **15** (1993), no. 4, 27–75.

[Wat08] Jon Watson, *Virtualbox: bits and bytes masquerading as machines*, Linux J. **2008** (2008), no. 166, 1.

[WLAG93] Robert Wahbe, Steven Lucco, Thomas E. Anderson, and Susan L. Graham, *Efficient software-based fault isolation*, In Proceedings of the 14th ACM Symposium on Operating Systems Principles, 1993, pp. 203–216.

[Wol96] George Woltman, *The great internet mersenne prime search*, 1996.

[WP05] R. Clint Whaley and Antoine Petitet, *Minimizing development and maintenance costs in supporting persistently optimized BLAS*, Software: Practice and Experience **35** (2005), no. 2, 101–121.

[WVB05] Peter H. Welch, Brian Vinter, and Frederick R.M. Barnes, *Initial experiences with occam-pi simulations of blood clotting on the minimum intrusion grid*, Proceedings of the 2005 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'05) (Las Vegas, Nevada, USA) (Hamid R. Arabnia, ed.), CSREA Press, June 2005, pp. 201–207.

[YS08] Bennet Yee and David Sehr, *Native client: A sandbox for portable, untrusted x86 native code*, Tech. report, 2008.

[ZBP05] Xin Zhao, Kevin Borders, and Atul Prakash, *Svgrid: a secure virtual environment for untrusted grid applications*, MGC '05: Proceedings of the 3rd international workshop on Middleware for grid computing (New York, NY, USA), ACM Press, 2005, pp. 1–6.

Appendix A

Publication 1

Proceedings of the 2006 International Conference on Grid Computing & Applications, GCA 2006, Las Vegas, Nevada, USA. CSREA Press 2006, ISBN 1-60132-014-0, pp. 121-126

Rasmus Andersen, Brian Vinter: Harvesting Idle Windows CPU Cycles for Grid Computing.

Harvesting Idle Windows CPU Cycles for Grid Computing

R. Andersen
Department of Computer Science
University of Copenhagen
Copenhagen, Denmark

B. Vinter
Department of Computer Science
University of Copenhagen
Copenhagen, Denmark

Abstract - *In this paper we demonstrate how to efficiently exploit the massive amount of idle CPU cycles from workstations and desktop PCs for Grid computing. The cycle harvesting is achieved by using sandbox technology and a generic guest operating system, specifically designed for the Grid resource. This relieves the resource owner from the complexity of installing and managing not only the resource software but also a large and diverse set of runtime environments that grid jobs may require. In addition, the sandbox provides a secure virtual environment, thus eliminating security issues for users and resource owners. Using this setup, resource owners can install a virtual machine of their own choice to provide a sandbox, and once the screen saver software has been downloaded, the resource is transparently and securely used as a Grid resource when the screen saver is activated.*

Keywords: Grid computing, screen saver science, sandboxing.

1 Introduction

Cycle scavenging, or Screen Saver Science, is an increasingly popular computing paradigm used within many fields of science that seek to tap the enormous amount of unused processing power from the millions of computers connected to the Internet. The paradigm is best known from the many successful Public Resource Computing projects, such as SETI@Home, where the idle time cycles are used for a dedicated scientific application. However, only a few attempts have been made to combine Screen Saver Science with Grid Computing in order to use idle cycles for any kind of application.

One of the Grid[1] promises is to make it possible to share and effectively use distributed resources on an unprecedented scale. Specifically, this includes harnessing the unused capacity of idle desktop PCs. Much research has been done to Grid-enable idle resources, yet no widely accepted system to effectively scavenge idle cycles, in particular idle Windows cycles, has been found.

The approach we take in this project is to use a virtual machine to provide a sandbox that completely separates the Grid job from the resource host system, so that, on the one hand, a grid job cannot compromise the host system, and on the other hand, the grid job is protected from other users of the system.

This paper addresses the problems that need to be solved in order to scavenge idle desktops for scientific use. First of all, a method to gain access to the CPU cycles on the idle resource must be found. To this aspect, security is a major issue. Ideally, a resource owner should neither install any software nor execute any foreign applications that, intentional or not, could compromise his system. Secondly, the resource, possibly hidden by a Network Address Translation router and a firewall must be attached to the Grid. Thirdly, it must be ensured that a given resource has installed the correct software base that a given Grid job requires. Finally, we introduce extra features to improve the model, for instance, a method to predict the idle time period of a resource in advance. Using this method, a job with a time deadline is submitted to a resource that is predicted to be available in the specified time frame.

The paper is organized as follows: Section 2 presents a generic approach to securely utilize the idle CPU cycles of many types of architectures. The means taken to Grid-enable the proposed model are discussed in Section 3. Section 4 addresses the problems as regards required runtime environments on the resources. A few MiG components that optimize the model are presented in Section 5, an experiment to test the model is carried out in Section 6, before we conclude in Section 7.

1.1 Related work

BOINC[2] is a software platform that allows many different distributed computing projects to utilize idle volunteered computer resources. Many Public Resource Computing systems use BOINC and research groups can with little effort create new projects. A project involves a set of applications that will be run in a BOINC client on a user's resource. As such, BOINC could be used for this project by running the proposed virtual machine as the project application.

A few projects have been found to combine Screen Saver Science with Grid computing, for instance the Entropia Virtual Machine[3], which is a commercial product, and

[4] that presents an extensive introduction to the approach of using virtual machines for Grid computing.

2 Sandboxing and cycle scavenging

The basic idea is to let resource owners install a so-called sandbox to provide a secure execution environment in which the Grid job is completely isolated from the host machine and vice versa. Such a sandbox may be in the shape of a virtual machine, which is exactly the approach that we have taken in this work. The alternative approach, which will not be investigated in this work, works by intercepting all operating system calls and inspect their validity.

Two techniques can be used to provide a virtual machine: Emulation and Virtualization[5]. Emulation provides the functionality of the target processor completely in software, which makes it a very secure approach. Also, the ability to emulate one processor type on any other processor type makes it ideal for this scenario. However, the method of interpreting the entire guest operating system, rather than running it on the native hardware, results in a significant performance drawback. When emulating a PC architecture on a PC, a compatibility layer that enables the target code to be run directly on the host processor, can reduce the performance penalty.

On the other hand, virtualization partitions hardware in multiple contexts, thus enabling running multiple operating systems on the same hardware resources simultaneously.

Several virtualization approaches exist[6]:

- Bare-metal Architecture
- Para-virtualization
- Full Virtualization, also known as Transparent Virtualization, or Hosted Architecture

The Bare-metal Architecture approach runs the guest operating system in *Ring 0*, the most privileged protection level in x86 architectures. Running multiple operating systems in the same protection level could potentially result in one of the systems compromising the other. Clearly, this approach is not acceptable for resource owners.

The Para-virtualization approach, known from Xen[16] et al., needs to modify the host system by interposing a *hypervisor* between the operating system and the hardware. The *hypervisor* then takes on the *Ring 0* and the operating system must be explicitly ported to run in *Ring 1*. These modifications to the host operating system exclude this approach.

The Full Virtualization approach has performance drawbacks, but is the least intrusive and thus chosen, not only because minimum intrusion is a goal of the MiG project, but also because the least intrusive approach means the highest number of potential participants. As shown in Figure 1, the virtual machine allows a guest operating system to run as an application in the host operating system. The virtual machine emulates the underlying hardware, thus creating a secure sandbox that allows an application written for one operating system, e.g. Linux, to be executed in another, e.g. Windows.

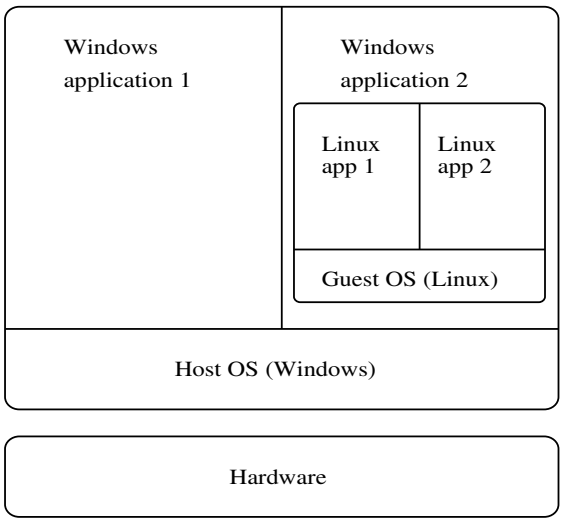


Figure 1: Full virtualization

The performance penalties are mitigated by kernel support that enables it to run most of the target application code directly on the host processor, thus achieving near native speed.

Regarding security, the virtual machine is a user space process that cannot do any harm to the host system as long as the permanent storage is protected properly. If the virtual machine is destroyed by a malicious application, the host system is not affected, and the virtual machine can start afresh.

The authors know of the following such solutions for the Microsoft Windows platform:

- VirtualPC[7]
- VMWare[8]
- Qemu + Qemu Accelerator Module[9]

Having downloaded and installed one of the virtual machines, a resource owner only needs to install a screen saver that starts the virtual machine upon activation and a tailor-made Linux image that is capable of running the Grid resource software automatically. In this manner, when the resource goes into screen saver mode, the virtual machine is activated and the Linux guest operating system is booted. The details of how to Grid-enable the hosted Linux system are explained next.

3 Enabling the sandbox for the Grid

The main problem with scavenging CPU-cycles from personal computers is that the vast majority are hidden behind a NAT router, i.e. they do not have global IP address and are therefore not reachable from the Internet.

Hence, to enable the sandbox for Grid Computing, care must be taken to circumvent the missing inbound Internet access. Naturally, this issue is highly dependent on the Grid middleware in question. In this work, the sandbox is enabled for the Minimum intrusion Grid, MiG, which is presented next, before the details of how to tailor the sandbox for MiG are explained.

3.1 Minimum intrusion Grid

MiG[10][11] is a stand-alone approach to Grid that does not depend on any existing systems, i.e. it is a completely new platform for Grid computing. The philosophy behind MiG is to provide a Grid infrastructure that imposes as few requirements on users and resources as possible.

The idea is to ensure that users only need a signed X.509 certificate, trusted by Grid, and a web browser that supports HTTP and HTTPS. A resource only needs to create a MiG user on the system and to support inbound ssh and outbound HTTPS. Initially, the resource must register to the MiG system using a certificate.

By keeping the Grid system disjoint from both users and resources, as shown in Figure 2, this model allows the Grid system to appear as a centralized black-box to both users and resources, and all upgrades and trouble shooting can be performed locally within the Grid without intervention from neither users nor resource administrators. Thus, all functionality is placed in a physical Grid system.

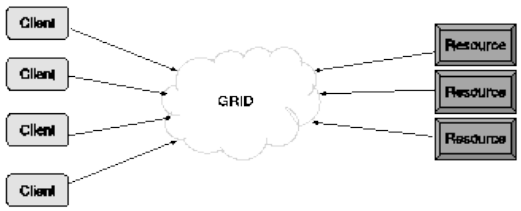


Figure 2: The abstract MiG model

The basic functionality in MiG starts with users submitting jobs to MiG and resources sending requests for jobs. A resource then receives an appropriate job from MiG, executes the job, and sends the result to MiG that can inform the user of the job completion. Thus, MiG provides full anonymity; users and resources interact only with MiG, never with each other.

3.2 The MiG Linux Image

As explained above, all that is required for a resource to join MiG, is to create a grid user account and support for incoming SSH and outgoing HTTPS.

So basically, the MiG Linux image can be built using any Linux distribution that runs an x86 system. Since the virtual machine provides a standardized virtualized set of hardware, compatibility amongst the wide range of different hardware setups on the resources will not be an issue.

The main concerns with respect to the distribution is the size and the start-up time. Both issues matter only for practical reasons, the size should be minimized to avoid an excessively large download, and, naturally, the start-up time should be minimized as much as possible.

As a base for the MiG Linux image we have chosen *ttylinux*[12], which is a minimalistic distribution that is easy to customize and, despite its very scarce space usage, provides an environment similar to larger distributions. The only shortcoming is the missing HTTPS support, so this had to installed manually.

In order to circumvent the missing inbound Internet access on resources that use NAT, it must be ensured that all communication is initiated by the resource. In MiG, this was easily integrated by small changes that only apply for sandboxes. In addition, it allows for directing jobs that users point out as Public Resource Computing jobs directly to a free sandbox.

The generic MiG Linux Image consists of a kernel and a Ram-disk that altogether take up less than 3 MB. An online generator modifies the generic image by giving it a unique resource name and a session id needed for requesting a job. Further, the resource owner can choose the size of a hard disk image file to provide as storage for the sandbox. Thus, certified resource owners can have a complete image built with a unique key allowing the sandbox to automatically request and execute Grid jobs.

In the standard MiG model, the identity of a resource requesting a job is verified by keeping the public SSH key of the resource in the MiG system and copying all job files to the resource over SSH. The sandbox model however, is modified to use a pull model on the resource where all files are transferred using HTTPS. Hence, a firewall in front of a resource only needs to be open for HTTP and HTTPS to allow the resource to run Grid jobs.

4 Runtime environments

Once the basic sandbox is in place it is possible to execute user applications within the virtual Linux machine. Many applications can be passed as executables from the Grid job and these need no further components to execute. Other, commonly used, applications may benefit from a preinstalled runtime environment, such as they are found on ordinary Grid resources.

Installing runtime-environments in the sandboxed environment could be done as on a conventional resource, which however would require the PC owners to personally maintain the sandboxed Linux distribution and this model is thus not desirable. Alternatively the sandbox image could be distributed with the initial Linux image, but this would greatly increase the size of the distribution image and in addition be a very static model.

The chosen solution allows individual research groups to maintain runtime environments for the sandboxed resources and at the same time allows the individual PC owners to control which runtime environments are downloaded and at which time. The runtime environments are kept in groups in virtual disk-partitions, in the form of a single file.

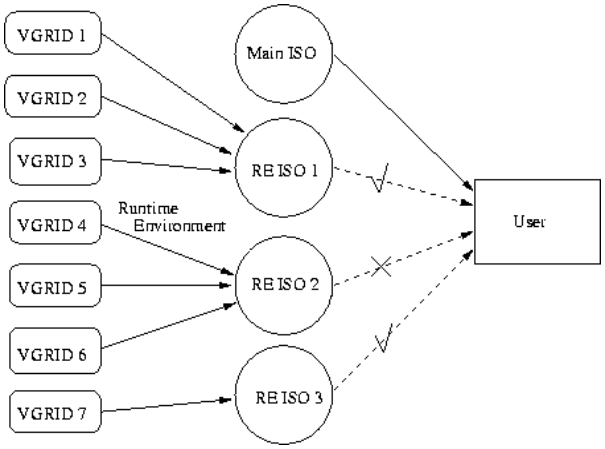


Figure 3: Runtime environments are maintained by the respective VGrids and grouped in virtual disks. Users are obliged to download the MiG Linux image, but can choose among the groups of runtime environments.

As shown in Figure 3, the PC owner can download a group of runtime environments from the VGrids, MiG's notion of a virtual organization, that maintain the runtime environments and when a job that uses a runtime environment is received by the sandboxed resource, the virtual Linux machine will mount the file system that contains the runtime environment. This way each runtime environment is kept isolated from the rest of the system, and can easily be built and maintained by the research groups that need them to be available for their executions.

5 MiG features

To improve and simplify the sandbox model further, MiG contains two components that apply: File access and scheduling.

5.1 Remote File Access

One difficulty that users report when using Grid is file access, since files that are used by Grid jobs must be explicitly uploaded to a Grid storage element and result files must also be downloaded explicitly. The MiG model introduces home catalogs for all Grid users, and all file references are relative to this home-catalog. This eliminates all naming problems, since MiG provides one simple access entry to a user's home-catalog. Furthermore, using the MiG Remote File Access library [13], a resource can, transparently and without recompiling or relinking applications, access application input and output files remotely, thus only downloading needed data and only uploading modified data.

5.2 Scheduling

Contrary to the majority of the existing Grid middlewares, where several levels of scheduling results in jobs being submitted to a resource where another level of scheduling takes place, MiG makes the scheduling for fairness much simpler as the local scheduling comes before the Grid scheduling. Thus, a single job is never left waiting a long time for CPU cycles once it has been submitted to a resource.

Existing Screen Saver Science systems all target problems that have many, usually millions, of independent tasks that often run for tens or hundreds of hours. Once a task has been assigned to a computer it will be processed while the computer is in screen saver mode. Processing is suspended if the screensaver is suspended and similarly resumed again along with the screensaver. An artefact of this model is that one never knows when the result of a given task is ready, and it is very hard to determine if the task has been lost or if it is simply only allowed to proceed very slowly.

For applications such as computational chemistry this model is very poorly suited. The number of tasks is usually in the tens or hundreds and it is often the case that analysis of the results can only start once the result of every task is in.

This is easily addressed by putting an upper time limit on each job, and if the time limit is exceeded, the job is resubmitted to another resource. However, in order to schedule a job with a deadline to a screen saver resource, we need to know how long the resource is availabe, i.e. how much time it takes before the screen saver is deactivated. To predict the available time slot of a screen saver resource, we use exponential average on an hourly

basis, which has proved to converge against the actual resource idle time quite fast.

6 Experiment

To verify our model, we have connected 8 sandbox ressources to MiG and submitted jobs that we pointed out as sandbox jobs. The 8 resources are all identical Windows PCs that host the virtual Linux machine that runs the Grid client code and includes the required runtime environment. The jobs are all NAMD[17] jobs, which is a software package for simulation of biomolecular systems.

Since we have 8 machines, we have chosen to submit 25 jobs. This means that there is 3 jobs for each machine and to avoid balanced execution, one last machine needs to execute one additional job before the experiment is completed. Further, using the 'minimization' option in NAMD effectively ensures different running times, thus ensuring unbalanced execution.

Running the jobs sequentially on one of the PCs results in a total running time of 2 hours, 25 minutes, and 22 seconds. When submitted to the Grid, the 25 jobs completed in 31 minutes and 45 seconds. Thus, despite the overhead, the model is deemed successful.

7 Conclusion

This work has shown how to eliminate the factors that have previously impeded the fusion of Public Resource Computing and Grid Computing to effectively utilize idle CPU cycles from desktop machines for any kind of Grid job.

The prohibiting factors include NAT-hidden resources, means to utilize Windows desktops, the workload required by a non-expert resource owner to install and manage all resource software, and the security issues involved with installing a large software base on the resource.

Using sandboxing technology and a generic Linux image, the Minimum intrusion Grid has successfully eliminated all of these limitations. Users need only download a bundle consisting of a screen saver, a virtual machine, and a special MiG Linux image in order to share their idle resources, whether they run Linux or Windows. The MiG system has proved flexible enough to easily deal with computers behind network address translators, and mobile processes and automatic resubmission of jobs solve the problem with resources that are cut off the network or leave the screen saver mode. Finally, the sandboxed environment ensures that the host system cannot be compromised.

Using this approach, a desktop computer volunteers as a Grid resource upon screen saver activation, and as soon as the screen saver is deactivated, the executing job either stops or migrates. Thus, the resource owner is completely unaffected by the Grid job.

8 References

[1]I. Foster, “The Grid: A New Infrastructure for 21st Century Science”, Physics Today, 55(2):42-47,2002

[2]http://boinc.berkely.edu

[3]B. Calder, A.A. Chien, J. Wang, D. Yang,"The Entropia Virtual Machine for Desktop Grids"

[4]R.J. Figueiredo, P.A. Dinda, J.A.B. Fortes, "A Case for Grid Computing on Virtual Machines"

[5]N. Kiyanclar, "A Survey of Virtualization Techniques Focusing on Secure On-Demand Cluster Computing"

[6]http://www.vmware.com/pdf/virtualization.pdf

[7]http://www.microsoft.com/windows/virtualpc/default.mspx

[8]http://www.vmware.com/products/player/

[9]http://fabrice.bellard.free.fr/qemu/

[10]B. Vinter ``The architecture of the Minimum intrusion Grid: MiG" In Communicating Process Architectures: pp. 189-201 Broenink J, Roebbers H, Sunter J, Welch P, Wood D (eds.) IOS Press, 2005

[11]Henrik Hoey Karlsen, Brian Vinter, "Minimum intrusion Grid - The Simple Model," wetice, pp. 305-310, 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05), 2005

[12]http://www.minimalinux.org/ttylinux/

[13]Rasmus Andersen, Brian Vinter, "Transparent Remote File Access in the Minimum Intrusion Grid," wetice, pp. 311-318, 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05), 2005

[14]W.L. George, J. Scott, "Screen Saver Science: Realizing Distributed Parallel Computing with Jini and JavaSpaces"

[15]M.J Litzkow, M. Livny, M.W. Mutka, "Condor-a hunter of idle workstations", Proc. of the 8th International Conference of Distributed Computing Systems, pp. 104-111, June, 1988

[16]P. Barham et al., "Xen and the Art of Virtualization", In Proc. SOSP 2003, Bolton Landing, New York, U.S.A. Oct 19-22, 2003

[17]James C. Phillips et al., “Scalable molecular dynamics with NAMD”, Journal of Computational Chemistry, 26:1781-1802, 2005

Appendix B

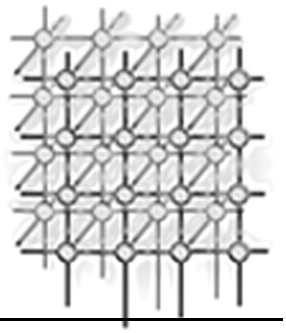
Publication 2

Concurrency and Computation: Practice & Experience, Volume 19, Issue 9 (June 2007), Second International Workshop on Emerging Technologies for Next-generation GRID (ETNGRID 2005), John Wiley and Sons Ltd, ISSN:1532-0626, pp. 1287-1298

Rasmus Andersen, Brian Vinter: Direct Application Access to Grid Storage

CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2005; **volume...** :1–15 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

Direct Application Access to Grid Storage



Rasmus Andersen*,†, Brian Vinter*

Department of Computer Science, University of Copenhagen, Denmark, Universitetsparken 1, DK-2100 Copenhagen, Denmark

SUMMARY

This paper describes the ideas behind and the implementation of a thin user-level layer to be installed on Grid resources. The layer fits in the Minimum intrusion Grid design by imposing as few requirements on the resource as possible and communicates with the server using only trusted and widely used protocols.

The model offers transparent, on-demand remote file access. By catching all application operations on files, these operations are directed towards the remote copy on the server, thus eliminating the need for transferring the complete file.

This implementation is targeted at the Minimum Intrusion Grid project, which strives for minimum intrusion on the resource executing a job. "Minimum intrusion" implies that a client need not install any dedicated Grid software. Hence, the proposed model is forced to use a user-level layer that automatically overrides the native I/O calls.

KEY WORDS: Grid Computing, Remote File Access, Minimum intrusion Grid

1. Introduction

Currently, while still in its early stages, the Grid[1] is primarily a domain for scientific applications. When such jobs are submitted to a commodity Grid, they are forwarded to a suitable resource that downloads all needed files and executes the job.

However, due to the well known fact that often only fragments of files are actually accessed and files for this kind of applications can be exceedingly large, we would be better off using a resource client that downloads only the needed data from the files in question and uploads only modified data.

In this paper, a layer providing on-demand remote file access is presented. This enables the resource to limit its file retrieval to a set of file fragments that hold the needed data, rather

*Correspondence to: Department of Computer Science, University of Copenhagen, Denmark, Universitetsparken 1, DK-2100 Copenhagen, Denmark
†E-mail: rasmus@diku.dk

than downloading the entire files. A natural extension to this model is to apply prefetching to increase performance as well as encryption to ensure optimal security during the transfer on the Internet and the processing on the resource.

The model is targeted at the Minimum intrusion Grid, MiG, which provides a Grid infrastructure with minimal requirements on the resource, but it is portable to other Grid solutions.

1.1. Minimum Intrusion Grid

The philosophy behind the MiG is to provide a Grid infrastructure that imposes as few requirements on users and resources as possible.

MiG is a stand-alone approach to Grid that does not depend on any existing systems, i.e. it is a completely new platform for Grid computing.

The idea is to ensure that users only need a signed X.509 certificate, trusted by Grid, and a web browser that supports HTTP and HTTPS. A resource on the other hand must allow inbound SSH connections and outbound HTTPS, and in addition create a local user, the Grid user, who can use secure shell, SSH, to enter the resource. All Grid jobs are executed by this dedicated user on behalf of the users that initially submitted the job. Finally, the resource owner must have a signed Grid certificate in order to register the resource.

By keeping the Grid system disjoint from both users and resources, as shown in Figure 1, this model allows the Grid system to appear as a centralized black-box to both users and resources, and all upgrades and trouble shooting can be performed locally within the Grid without intervention from neither users nor resource administrators. Thus, all functionality is placed in a physical Grid system. A more detailed description of the architecture and the functionality can be found in [2] and [3].

The challenge is to make the desire for minimum intrusion coexist with a large set of features, including scalability, autonomity, i.e. updating grid without causing users and resources inconvenience, anonymity, i.e. users and resources are anonymous to each other, fault tolerance, firewall compliance, etc.

In order for this client to comply with MiG, it must be ensured that it is entirely user-level and is installable without administrator privileges but still automatically overrides the native file system routines. Thus user applications need not be recompiled or rewritten using a custom MiG API.

1.1.1. File access in MiG

One difficulty that users report when using Grid is file access, since files that are used by Grid jobs must be explicitly uploaded to a Grid storage element and result files must also be downloaded explicitly. The MiG model introduces home-catalogs for all Grid users, and all file references are relative to this home-catalog. This eliminates all naming problems when implementing the proposed access layer, since MiG provides one simple access entry to a user's home-catalog.

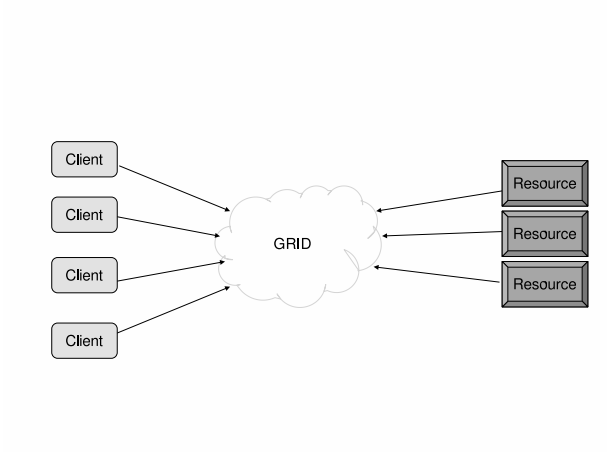


Figure 1. The abstract MiG model

1.2. Motivation

Job submissions in most Grids follow the intuitive flow: the user submits a job that the Grid forwards to a resource that downloads the executable along with the input files, the resource executes the job, and eventually the outputfiles are uploaded to the file server.

Often, only the first part or scattered fragments of input files are really needed. For Grid resources, this results in wasting storage and bandwidth by downloading entire files when only small pieces are needed.

Furthermore, lots of valuable time is lost as the job execution is delayed until everything has been downloaded. Of course, with huge input files the problem is more evident, and will be a limiting factor of performance, since the download time becomes a significant part of the total time from job submission to job termination. Similarly, the outputfiles may equally well be partial. This suggests the need for a resource model which automatically downloads only the needed data and uploads only modified data.

This paper describes the problems we face when designing such a layer for MiG in section 2. The implementation of the corresponding solutions is given in section 3. Section 4 shows the performance of the client before we conclude in section 5.

1.3. Related Work

Numerous systems, such as NFS and AFS, provide transparent access to remote files. Only a few run entirely in user space, for instance FUSE, Filesystem in User Space, and LUFS, Linux Userland FileSystem. LUFS enables mounting of a number of file systems including SshFS, thus allowing a user to mount files accessible by ssh in the file hierarchy. However, since these

systems require root privileges to install a kernel module on the resource, they don't fit in the MiG model.

The Ufo Global File System[4] supports extending or altering the functionality of certain system calls by intercepting them in a user level module. The interception is achieved by standard tracing facilities. This strategy avoids the need for recompilation, relinking and administrator privileges. Using Ufo, one can transparently access personal accounts at remote sites using different protocols. The only drawback to this model is the interception method, which is quite expensive. Ufo is a great alternative for applications that issue a small number of system calls, but this is not the case for the Grid jobs in this project.

The ORFA client[5] provides efficient access to remote file systems using a preloaded light-weight shared library that overrides standard file access routines. The file management is handled by virtual file desriptors that enable remote files to be accessed and manipulated as local files. The exact same approach is used in this project. However, neither a system providing on-demand file access nor a protocol supporting reading and writing ranges of bytes has been found.

2. Design

Providing on-demand transparent remote file access for a resource in a Grid environment first of all requires a protocol that supports retrieval or sending of randomly requested or modified data. Next, we have to make sure that certain file access routines on the resource are automatically overridden and redirected to the file on the server. Finally, since the local file management is overruled, a local file management mechanism to ensure correct file access behaviour is needed.

2.1. File Data Transfer

HTTP/1.1 supports transferring ranges of bytes from a file in `GET` requests, but there is no range parameter in `PUT` requests, which is needed for remote writing. Since no file transfer protocol has been found to support range requests for both `PUT` and `GET`, a custom protocol is developed. The protocol should also support `OPEN` and `CLOSE` requests. Due to the MiG restriction of firewall compliance, the connection is encapsulated in an SSH-tunnel.

2.2. Catching Access to Job Files

The basic idea of the layer is to make user applications think the job files really exist in their entirety on the resource, yet they only exist on the server. This is achieved by interposing a user space file access layer between the application and the operating system. The purpose of this layer is to catch access to job files and direct them to the server holding the files.

Figure 2 shows the file access model. All file access routines issued from the user application are handled by the MiG file access layer that sends a request to the server (1). The server then replies (2) and an appropriate action is taken by the file access layer before the application

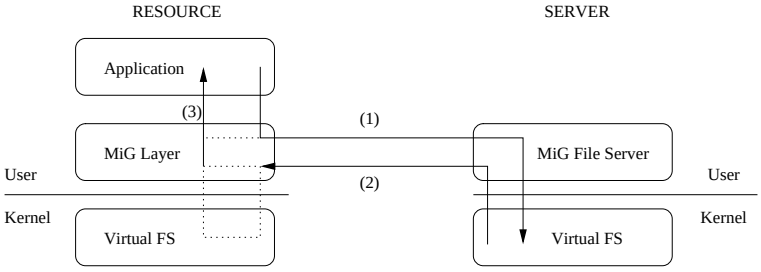


Figure 2. Overview of the MiG File Access Model

receives the result (3). The dotted lines on the resource depict local management of remote files with and without server and kernel intervention, explained in 2.3.

Access to files not mentioned as job files are forwarded to the native file system.

Reading a block from an input file would result in a server request for the specified data that would be delivered in the buffer supplied by the application. The resource executing the job, would only have to download the executable before initiating the job. Subsequent file accesses result in fetching only the requested data and performing operations conforming to POSIX behavior on the fragment of the remote file.

2.3. File Handling

A shortcut to overriding the complete set of file manipulating routines is taken by actually creating the remote input file on the resource. Besides the obvious advantage of not implementing all file access routines, including maintenance of their evolution, we also avoid implementing complex UNIX functionalities. Instead, the input file is created on the resource and before any file access, it is ensured that accessed data is available.

Thus, as the dotted line on figure 2 illustrates, some remote file accesses, i.e. `open`, upon server response result in requesting the resource kernel system to create the file, whereas other calls, i.e. `read`, remain in the MiG layer on receipt of the server response and returns data directly to the application. Had the requested data already been fetched, it would have been returned immediately without server or kernel intervention.

Table I shows the set of file access routines that are being overridden.

3. Implementation

The idea is to make the MiG file access layer do all file management, i.e. maintaining a file pointer and checking file access etc. This is achieved by keeping a user-level structure, that upon an `open` call gathers all information about the file. The layer then uses the real `open` call to create the file locally and maps that file into memory in its entirety. The file descriptor

Table I. Overridden file access routines.

Description	Functions
File Access	open, close, read, write (and similar stream functions)
File position	lseek, fseek, ftell, fgetpos, fsetpos, rewind
Synchronization	fsync, fdatasync, msync
Memory mapping	mmap, munmap, mremap
Memory protection	mprotect

returned from the `open` call is finally given to the application. Creating and mapping the file in its entirety does not waste memory, since nothing is allocated until it is needed.

3.1. Virtual File Descriptors

All input files are described by virtual file descriptors. The virtual file descriptor is a structure containing several pieces of information about the file: the local file descriptor that is returned to the application, the socket descriptor, the length of the file, a pointer to the location in memory where the file resides, the file name, flags indicating access mode, etc.

Figure 3 shows how the layer interacts with the descriptor management in the kernel I/O subsystem. The figure also depicts the mapping of an input file in a snapshot where two fragments of the file have been accessed, and thus retrieved from the file server, see section 3.2.

3.2. Memory Mapping of Input Files

All remote input files opened by the user application are created on the resource and mapped into memory. Although the file is initially empty, it is mapped in its full length. Subsequent `read` calls will get data from the server and fill in requested pieces.

As shown in figure 3, the memory mapped file is highly fragmented, because only the needed data is retrieved, while only used space is actually allocated. Accesses to empty memory addresses within the scope of the file thus result in segmentation faults. Hence, a procedure to handle segmentation faults must be introduced. This is implemented using the `sigaction` system call that invokes a procedure upon receipt of a `SIGSEGV` signal. This procedure gets the faulting address, determines the file owning the address, translates the address into a file offset, and sends a request to the server for the block surrounding the offset.

Mapping input files into memory has several advantages:

- The layer only deals with memory addresses when accessing files.
- Direct memory access; the layer reads data from the socket directly into the correct location in memory. This prevents copying from a buffer.

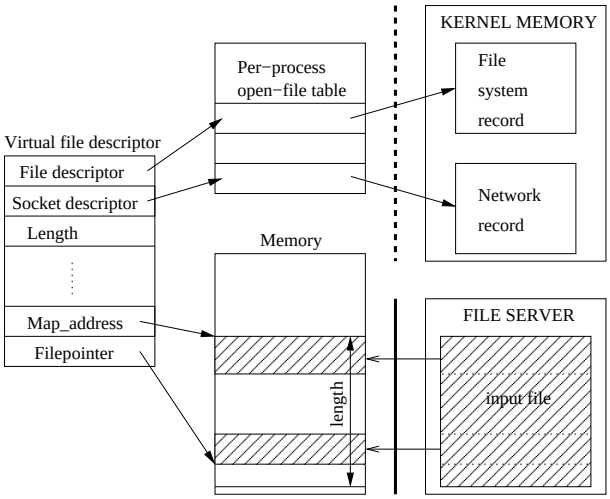


Figure 3. Interaction with the I/O subsystem.

- Reading prefetched, cached or already copied data is returned immediately without a system call.
- The memory mapped image may be returned directly to the application if it issues an `mmap` call.

Any negative side-effects, such as excessive page-swapping due to the larger active address space, is still to be demonstrated before analyses can be made.

3.3. File Access on the Resource

Linux supports preloading of user defined libraries to override a subset of the standard I/O functions. By setting the LD_PRELOAD environment variable, the linker first checks the preloaded library for a matching symbol name. Thus, one can write a private set of I/O functions, while functions not overridden are automatically handled by the native system. This feature requires user applications to be dynamically linked, which most frequently is the case. Statically linked applications are not supported by this model.

If the application creates a file that is not declared in the job script, the layer should forward the creation request to the native I/O system. Since the `open` call is overridden, the linker must be instructed to search for the next matching symbol. This is done using the *dlfcn* library that provides a handle, `RTLD_NEXT`, for finding the next occurrence of a function in the original search order after the current library.

Naturally, all functions that are overridden must exhibit the same behaviour as the corresponding system function and conform to the ANSI C standard.

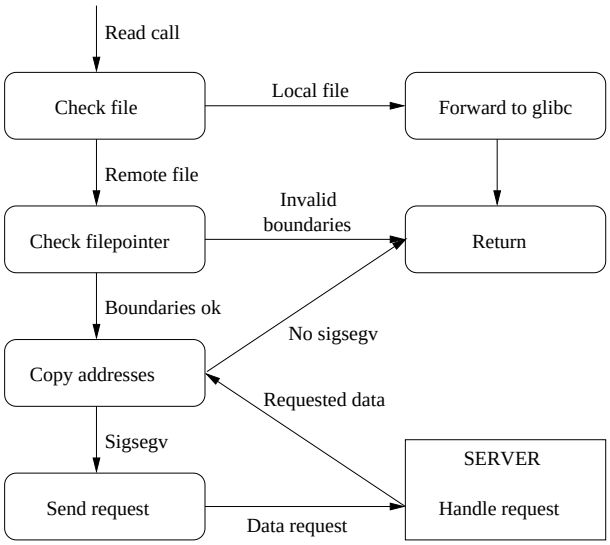


Figure 4. Data flow diagram for read calls.

Figure 4 shows the data flow for a `read` call. First, the layer determines whether the file is a job file or not. In the former case the call is forwarded to the native I/O system, in the latter the virtual file descriptor is retrieved. Then, the boundaries of the file are checked to avoid unnecessary requests.

Next, the layer copies data directly from the file pointer to the user buffer. As explained in section 3.2, this may raise a `SIGSEGV` that invokes the procedure to get the requested data. If the application reads a chunk larger than the network data-transfer size, the copying will continue raising `SIGSEGV`, which causes further data to be transferred. Eventually all data requested in the `read` call is fetched and the function can return. Hence the cycle in the figure.

Writing to a file results in sending the content of the supplied buffer to the server that writes to the real file. If the file is open for both reading and writing, the data is also copied to the location in memory pointed to by the file pointer. Seeking in a file moves the file pointer.

If the user calls `mmap` on an input file, and the access modes for the call match the open mode of the file, the layer just returns the address at which it mapped the file itself. Closing a file results in synchronizing the file to disk, unmapping it, calling the real `close` on the file descriptor, and closing the socket.

3.4. Prefetching and Block Size Granularity

One can never tell if user applications read reasonable amounts of data; one might read in chunks of 100 bytes, another in chunks of 1k bytes. Clearly, sending a network request for such

data amounts is very inefficient. For this reason, the layer always fetches blocks of multiples of 4k. This increases network performance and eases the file management due to the alignment of the blocks in memory.

Thus, if the user application requests bytes 1024 to 2047, the layer sends a request for the block surrounding the specified range, i.e. a request for bytes 0-4095, for a block size of 4k.

The choice of the block size depends only on the user application. If the application reads all blocks sequentially without much data processing, a large block size is preferred. The file access layer would then be able to continuously read the next block, thus always reading a block ahead and hiding the latency.

However, if the application only reads small scattered fragments of the file, reading ahead is useless and the application would have to wait for at large block to arrive. Thus, in this scenario a small block is preferred.

Since it is impossible to tell the nature of user applications in advance, we need to make the block size adapt dynamically to the application, based on a 1-block read-ahead prefetching algorithm.

Every time the file access layer receives a block from the server, a thread is started to fetch the next block. The next time the application calls `read`, it is noted whether the prefetching is finished and whether the prefetched block is the one we need now. Based on these observations, the blocksize either increases, decreases or remains unchanged.

3.5. Security

In this project, only security issues regarding job files are considered. Executing a job on a foreign resource will always suffer from the inability to completely elude the resource administrator from surveying, copying or deleting job files. If the job files are highly confidential, we can never guarantee that the resource administrator cannot gain access to them. However, we can make it very hard to intrude on them.

First of all, everything is transferred on the Internet in SSH-tunnels. Secondly, in scenarios where optimal security is required, a user can choose to have the files block-encrypted on the file server. The file access layer then defers decrypting to the moment before copying to the user buffer. After the copying, it is reencrypted.

Of course, the data now lies unencrypted in the user buffer, but it is now the responsibility of the user.

4. Performance

The performance of the proposed model is measured by some experiments that cover the access patterns that apply to most applications. The results are then compared to, on the one hand, local execution, where the application and the files are co-located, and on the other, a standard Grid model where all files are downloaded prior to the job execution.

When comparing to a standard Grid model, we choose to compare to the performance of a basic transfer method, *curl*[6]. This way any other project may perform the same experiment

and by translation compare their performance to that of this model. Such systems could include bbftp[7] and UDT[8].

The prefetcher and the dynamically adjusting block size are introduced in the model solely to improve performance. The impact of these mechanisms are studied lastly.

4.1. Experiments

The model is tested in 4 scenarios. Firstly, we investigate the basic overhead, secondly the performance in an I/O limited application is examined. Thirdly an I/O balanced is analyzed, and finally we test the model in a scenario where only a small portion of a huge file is used.

In order to determine the basic overhead of the remote access protocol, the first experiment simply reads a 1-byte file and verifies the content. This experiment provides us with a baseline for the performance of the remote access model.

In the second test, the application checksums a 1 GB file. The calculations that are involved are so few and simple that the application is simply limited by I/O performance. Naturally, the latency of getting a new page from disk is less than retrieving the data through the remote access layer. Thus, this is the kind of application that does not really perform well on Grid. Still, the copy-semantics should be faster than the remote access layer, since bulk transfer of a complete file is more efficient than the blocked access model.

In the third experiment, a 1GB input file, which requires some processing from the application is traversed. The input file contains a series of numbers that the application reads and then computes a corresponding fibonacci value on one of the numbers. This test will show the benefit of using prefetching in combination with starting the job immediately without waiting for the input file to arrive and should prove favorable to the remote access model.

Finally, a large file containing a B+ tree of order 4 is searched using a random key. The test is run with 10 different keys. A B+ tree is an ideal structure for the remote access model since it is designed to branch out in a large number of directions and to contain a lot of keys in each node. This ensures that the height of the tree is relatively small. Thus, only a small number of nodes, one in each level of the tree, must be read to retrieve an item.

4.2. Results

Table II shows the results of the 4 experiments. The results of using the proposed layer, shown in column 4, are compared to a model that just downloads the entire input file and executes the job, column 3. Local execution with local input files is shown in column 2. All experiments are performed on the same dedicated resource and server, both 2.4 GHz Intel Celeron with 512 MB RAM, using a 100Mbps network connection.

As the result of reading a 1 byte file reveals, using the proposed remote file access layer does not incur much overhead compared to the local execution. This experiment also shows that the layer is faster than using external *curl* to download the file.

The result of the I/O intensive application is surprising since bulk transfer is faster than requesting the entire file block-wise. The reason why the remote file access model is faster is due to a combination of the prefetcher and the server. The prefetcher keeps increasing the

Table II. Result of experiments in seconds.

Experiment	Local	Copy	Remote
1 B file	0.0002	0.1520	0.0080
Checksum	50.1100	130.1000	114.0300
Fibonacci	632.8300	721.2200	600.7200
B+ tree	0.0002	30.6920	0.0186

block size, because it detects sequential access, and the python server is actually faster than the dedicated apache server that *curl* consults.

The result of the I/O balanced application, the Fibonacci experiment, shows that the layer is able to hide the transfer time during data processing because it is faster than the download model. Amazingly, running this application using the proposed layer is faster than local execution. This is due to a combination of perfect prefetching and the design shown in figure 2. Almost all read requests are returned immediately by the file access layer without server or kernel intervention. Thus, all requested blocks are already in memory prior to the read calls. During local execution, all blocks are fetched using a system call and expensive disk access.

The B+ tree experiment performs excellently on the proposed model. The depth of the tree is 9, hence only 9 blocks are fetched plus an additional header block and wasted blocks from the prefetcher. Often the size of the B+ tree file is much larger than this one (357 MB), which in effect prohibits this kind of applications from execution with Grid implementations that use the download model. The speedup between the download model and the proposed model is a factor of 1650. Naturally, a native execution baseline is much more efficient at this stage, however overheads less than a second should be acceptable for any job one may choose to submit to Grid.

4.3. Performance in a heterogeneous network

The ability to mitigate the penalties incurred by heterogeneous network conditions is best illustrated by the following figures that show the impact on the execution time as the latency between the resource and the server increases. The latencies to different university centers are simulated on the file server by inserting a sleep call between the request and the response.

Figure 5 shows the overhead and b-tree applications. Since the overhead application only reads 1 byte, the effects of increased latency are noticeable but insignificant.

Since the B+ tree application reads a fixed number of scattered blocks from the file, the latency is added to each block transfer time.

In Figure 6 the benefit of a dynamic block size is apparent: The running time of the fibonacci experiment is unaffected by the increased latency. As the latency increases, the library fetches

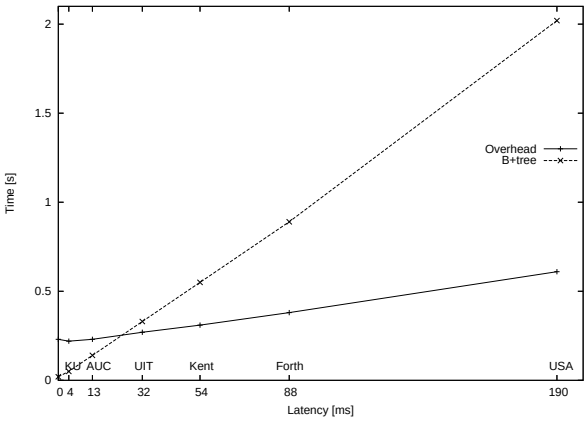


Figure 5. Results of the overhead and B+tree applications

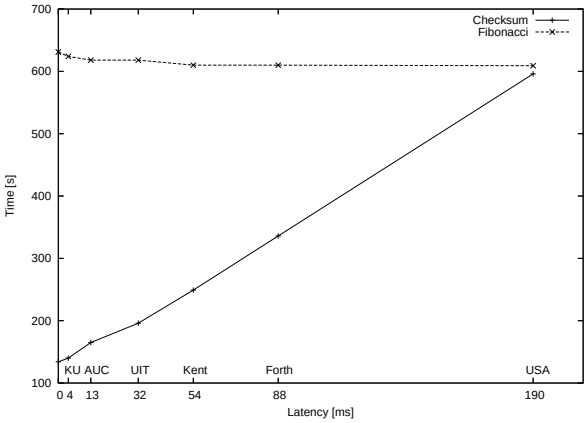


Figure 6. Results of the checksum and fibonacci applications

larger blocks, thus achieving a larger bandwidth and providing more data for processing in each request.

The checksum application is more affected by the increased latency, because the application is I/O bound.

5. Conclusion

The MiG project aims at providing a new stand-alone Grid infrastructure that imposes as few requirements on users and resources as possible. The on-demand transparent remote file access layer is specifically designed towards this project without compromising neither the design requirements nor the features of this Grid project.

Using this user-level layer, it has been shown that a resource does not need to neither download entire input files before job execution nor upload output files after the job has terminated. Instead it just starts the job, downloads portions of the input file when needed, and uploads modified data when written. This functionality is achieved without requiring user application to use special API's or to be recompiled.

The results show that this model provides access to the Grid for a whole niche of applications that were previously impeded by unnecessary transfer of an enormous amount of data, namely applications using partial file traversal on huge input files, such as B+ trees.

This is also the case for applications such as high-energy physics applications that analyze relatively small fragments of massive physics database files.

REFERENCES

1. Foster I. A new infrastructure for 21st century science. In *Physics Today* 2002 **55**(2):42–47.
2. Vinter B. The architecture of the Minimum intrusion Grid: MiG. In *Communicating Process Architectures*: 189–201 Broenink J, Roebbers H, Sunter J, Welch P, Wood D (eds.) IOS Press, 2005;
3. Karlsen HH, Vinter B. Minimum intrusion Grid: the simple model. In *Proceedings of the fourteenth IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2005)*:305–310 IEEE Computer Society, 2005;
4. Alexandrov AD, Ibel M, Schauser KE, Scheiman CJ. Extending the Operating System at the User Level: the UFO Global File System. In *1997 Annual Technical Conference on UNIX and Advanced Computing Systems (USENIX'97)*: 77–99
5. Goglin B, Prylli L. Transparent remote file access through a shared library client. In *Proceedings of the 2004 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2004)*:1131–1137 CSREA Press, 2004
6. Curl - transfer a URL, http://curl.haxx.se/, 2006-09-07
7. bbftp, large files transfer protocol, http://doc.in2p3.fr/bbftp/, 2006-09-07
8. UDT: UDP-based Data Transfer Protocol, http://udt.sourceforge.net/, 2006-09-07

Appendix C

Publication 3

Proceedings of the Sixth International Conference on Engineering Computational Technology, ECT 2008, Athens, Greece. B.H.V. Topping (Editor), Civil-Comp Press, Stirling, United Kingdom, ISBN 978-1-905088-26-3, paper 5, 2008. **R. Andersen, B. Vinter, D.A. Power and J.P. Morrison: Supporting MiG and WebCom Interaction**

Supporting MiG & WebCom Interaction

R. Andersen[1], B. Vinter[1], D.A. Power[2] and J.P. Morrison[2]

[1]eScience Centre, University of Copenhagen,
Copenhagen, Denmark
[2]Department of Computer Science, University College Cork,
Cork, Ireland

Keywords: MiG, WebCom, sandbox, workflow, grid, PRC Model

The Minimum intrusion Grid (MiG) [1] and WebCom [2] are two separate and independent middleware implementations, developed at different universities for different target groups. Both systems implement grid middleware properties, and provide a fully functional grid system, yet they are fundamentally different by design and implementation. Despite their differences, both middleware solutions were designed for user transparency with the vision of the ability to facilitate a grid operating system that would fully leverage the grid potential.

This report presents a collaborative effort between these two grid middleware systems, where we present a methodology that on the one hand provides WebCom users access to special sandboxed environments on a computing platform composed of the resource richness of the PRC model, and on the other hand, MiG users can exploit the ease of use of the Visual programming model offered by WebCom by embedding their applications within a dynamic workflow execution environment. This also eases utilization of the MiG sandboxes significantly thus effectively expanding the user group of the MiG system.

The PRC model has shown its potential in many scenarios; many scientific projects utilize a PRC platform, and many private people are willing to contribute. Equally high on the hype curve is the trend of visually aided application development. Each of the two middlewares are specialized in each of these domains, which makes a combined effort quite unique. Ultimately, the potential of this work allows people with no or only little knowledge about programming and Grid technology to easily make use of the enormous pool of aggregated compute resources in the Grid.

Using sandbox technology, the resource platform is rendered uniform, and applications are securely isolated from other processes on the host system and vice verse. The applications are constructed using Condensed Graphs where the traditional complexities involved in developing parallel and distributed applications are removed from the user burden.

The report details initial investigations into these interactions between the two quite different grid middlewares, by adding functionality to the WebCom that facilitate the

targeting and execution of MiG jobs. A test application proved successful and justified the need for further research to be conducted; many different scenarios seem ideal for this collaboration, including the use of more scientific computations such as Fast Fourier Transformations and extending the nodes available when creating visual applications for MiG.

References

[1] B. Vinter: The Architecture of the Minimum intrusion Grid: MiG. Proceedings of Communicating Process Architectures 2005

[2] J.P. Morrison, B. Clayton, D. Power, A. Patil: WebCom-G: Grid Enabled Metacomputing- The Journal of Neural, Parallel and Scientific Computation. Special Issue on Grid Computing. Editors H.R. Arabnia, G.A. Gravvanis, M.P. Bekakos, Vol. 12(3), PP. 419-438, September, 2004

Abstract

This paper presents a unique collaboration between two grid middleware systems: the Minimum Intrusion Grid (MiG) and WebCom. Although both systems implement grid middleware properties, and provide a fully functional grid system, they are fundamentally different by design and implementation, and target different end-users.

A strategy that is mutually beneficial is obtained by supporting MIG and WebCom interactions that on the one hand provides WebCom users access to special sandboxed environments on a computing platform composed of the resource richness of the PRC model. On the other hand, MIG users can exploit the ease of use of the Visual programming model offered by the dynamic workflow execution environments offered by WebCom.

Keywords: MiG, WebCom, sandbox, workflow, grid, PRC Model

1 Introduction

When first introduced, one of the big selling points of Grid Computing [1], was to make distributed computing resources as easily accessible as electricity in a power grid. Today, more than a decade later, the prohibitive barrier between grid application developers and efficient exploitation of the huge pool of aggregated resources still exists; developers still face several challenges when attempting to interact and utilize the Grid and a deeper insight to the underlying technologies is required. Providing uniform access to distributed heterogeneous resources to end users in the same manner that we get electricity from a socket in the wall remains an illusion.

Many Grid solutions exist, most of them are either built upon, or direct descendants of the Globus toolkit [2]. These systems have several shortcomings [3] that would greatly complicate a solution similar to the methodology presented here, most

importantly their lack of strong scheduling, poor scalability, firewall dependencies on connected resources and client, and their big resource and client software packages.

Gaining widespread acceptance as an important computing platform depends on making the technology accessible to general exploitation, not only for grid developers but also for non specialists. In practice, hiding the underlying complexities of a Grid system not only requires grid applications to be freed from all architectural details of the computational resources connected to the Grid, but also the set of tools needed to develop the applications and interact with the Grid system.

The Minimum intrusion Grid (MiG) [3] and WebCom [4] are two separate and independent middleware implementations, developed at different universities for different target groups. Both systems implement grid middleware properties, and provide a fully functional grid system, yet they are fundamentally different by design and implementation. Despite their differences, both middleware solutions were designed for user transparency with the vision of the ability to facilitate a grid operating system that would fully leverage the grid potential. This report presents a collaborative effort between these two grid middleware systems.

MiG strives for lowering the entry cost for users and resource providers by moving as much functionality as possible into a fat, centralized black-box grid system. MiG leverages the concept of 'sandboxes' to provide a non-intrusive and highly dynamic Grid computing infrastructure. Once a sandbox job is submitted to MiG, a sandbox is instantiated on a resource providers hardware. This sandbox then fetches and executes the application. Once this sandbox has completed its task, results are returned to MiG, and the sandbox is subsequently terminated. Thus every application in such a sandbox is guaranteed to execute in a fresh environment. This facilitates a great degree of non-intrusiveness on host resources and has proved to close the gap between Grid Computing and Public Resource Computing (PRC) models.

WebCom focuses on providing a suite of tools encompassing application development through to execution. Applications are expressed as workflows constructed from tasks of varying complexities such as grain-size, target execution environments and resource requirements. These applications are constructed from a 'palette' of services and submitted to a WebCom instance for execution. This mechanism allows for users with little knowledge of programming to create and execute applications of varying complexity on a vast range of distributed computing architectures from Networks of Workstations, to Clusters, to Grid computing infrastructures.

Whilst it is clear that both systems target different audiences and application execution models, a strategy that is mutually beneficial is obtained by supporting MiG and WebCom interactions. Both middlewares are highly capable of providing specialised features unique to their own domain. However, by utilising these features, we present a methodology that on the one hand provides WebCom users access to special sandboxed environments on a computing platform composed of the resource richness of the PRC model, and on the other hand, MiG users can exploit the ease of use of the Visual programming model offered by WebCom by embedding their applications within a dynamic workflow execution environment. This also eases utilization of the

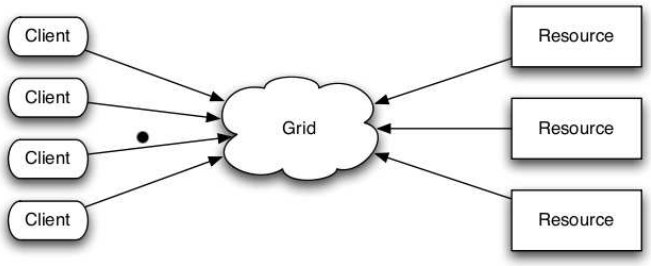


Figure 1: The abstract MiG model.

MiG sandboxes significantly thus effectively expanding the user group of the MiG system.

The following section gives a deeper introduction to each of the two middleware systems. Next, Section 3 describes in technical details the implementation of the presented approach. A motivating test application is presented in Section 4, and the report is closed by conclusions and future work in Section 5.

2 Grid Systems Introduction

2.1 Minimum intrusion Grid

MiG is a stand-alone approach to Grid that does not depend on any existing systems, i.e. it is a completely new platform for Grid computing. The philosophy behind the MiG is to provide a Grid infrastructure that imposes as few requirements on users and resources as possible. The overall goal is to ensure that users only need a signed X.509 certificate, trusted by Grid, and a web browser that supports HTTP and HTTPS. A fully functional resource only needs to create an MiG user on the system and to support inbound ssh and outbound HTTPS.

By keeping the Grid system disjoint from both users and resources, as shown in Figure 1, this model allows the Grid system to appear as a centralized black-box to both users and resources, and all upgrades and trouble shooting can be performed locally within the Grid without intervention from neither users nor resource administrators. Thus, all functionality is placed in a physical Grid system.

The basic functionality in MiG starts with users submitting jobs to MiG and resources sending requests for jobs. A resource then receives an appropriate job from MiG, executes the job, and sends the result to MiG that can inform the user of the job completion. Thus, MiG provides full anonymity; users and resources interact only with MiG, never with each other.

2.2 MiG Sandboxes

Typically, the resource farm of a grid system is comprised of unix/linux computers from many different compute centres. While the computational power provided by these may be significant, it is in no way comparable to the increasingly popular Public Resource Computing (PRC) models, such as BOINC [6], a middleware for volunteer computing[1]. Disguised as a screen saver, the PRC client software harnesses the idle time CPU cycles from millions of privately owned Internet-connected PCs across the globe.

However, there is a huge gap between such systems and Grid Computing in that the entry cost in terms of workload for application developers is far from the vision of a grid operating system[2]. In order to close this gap, MiG introduced sandboxes capable of tapping the enormous amount of unused processing power from standard desktop computers. First and foremost, minimizing the workload required by resource owners for installing and managing a Grid resource, and ensuring the integrity of the donated resource is utterly important. Therefore, all grid application are sandboxed in a virtualized environment.

Due to the renewed popularity of virtualization over the last few years, virtual machines are being developed for numerous purposes and therefore exist in many designs, each of them in many variants with individual characteristics. Despite the variety of designs, the underlying technology encompasses a number of properties beneficial for Grid Computing [7]:

Platform Independence: Since moving application code around as freely as application data is an intrinsic part of a grid system, it is highly profitable to enable applications to be executed anywhere in the grid. Virtual machines bridge the architectural boundaries of computational elements in a grid by raising the level of abstraction of a computer system, thus providing a uniform way for applications to interact with the system. Given a common virtual workspace environment, grid users are provided with a compile-once-run-anywhere solution.

Furthermore, a running virtual machine is not tied to a specific physical resource; it can be suspended, migrated to another resource and resumed from where it was suspended.

Host Security: To fully leverage the computational power of a grid platform, security is just as important as application portability. Today, most grid systems enforce security by means of user and resource authentication, a secure communication channel between them, and authorization in various forms. However, once access and authorization is granted, securing the host system from the application is left to the

[1] In 2006, the BOINC projects reached 400 TFlop/s, while BlueGene/L topped the top 500 list of supercomputers with 280 TFlop/s

[2] For instance, according to the BOINC website, it takes three man-months to port an existing application to the BOINC framework and about $5000 for hardware used for project maintenance

operating system.

Ideally, rather than handling the problems after system damage has occurred, harmful - intentional or not - grid applications should not be able to compromise a grid resource in the first place.

Virtual machines provide stronger security mechanisms than conventional operating systems, in that a malicious process running in an instance of a virtual machine is only capable of destroying the environment in which it runs, i.e. the virtual machine.

Application Security: Conversely to disallowing host system damage, other processes, local or running in other virtualized environments, should not be able to compromise the integrity of the processes in the virtual machine.

System resources, for instance the CPU and memory, of a virtual machine are always mapped to underlying physical resources by the virtualization software. The real resources are then multiplexed between any number of virtualized systems, giving the impression to each of the systems that they have exclusive access to a dedicated physical resource. Thus, grid jobs running in a virtual machine are isolated from other grid jobs running simultaneously in other virtual machines on the same host as well as possible local users of the resources.

Resource Management and Control: Virtual machines enable increased flexibility for resource management and control in terms of resource usage and site administration. First of all, the middleware code necessary for interacting with the Grid can be incorporated in the virtual machine, thus relieving the resource owner from installing and managing the grid software. Secondly, usage of physical resources like memory, disk, and CPU usage of a process is easily controlled with a virtual machine.

Performance: As a virtual machine architecture interposes a software layer between the traditional hardware and software layers, in which a possibly different instruction set is implemented and translated to the underlying native instruction set, performance is typically lost during the translation phase. Despite of recent advances in new virtualization and translation techniques, and the introduction of hardware-assisted capabilities, virtual machines usually introduce performance overhead and the goal remains achieving near-native performance only. The impact depends on system characteristics and the applications intended to run in the machine.

To summarize, virtual machines are an appealing technology for Grid Computing because they solve the conflict between the grid users at the one end of the system and resource providers at the other end. Grid users want exclusive access to as many resources as possible, as much control as possible, secure execution of their applications, and they want to use certain software and hardware setups.

At the other end, introducing virtual machines on resources enables resource owners to service several users at once, to isolate each application execution from other users of the system and from the host system itself, to provide a uniform execution

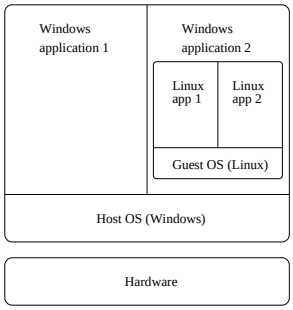


Figure 2: Full virtualization; applications running in the guest OS in the virtual machine are isolated and can only compromise the VM, not the host system

environment, and managed code is easily incorporated in the virtual machine.

2.3 The MiG Sandboxes

Taking advantage of these virtualization properties, the MiG sandbox solution effectively combines Grid Computing and PRC computing. Contributors download a screen saver, a virtual machine of their own choice, and a custom made MiG Linux image. Upon screen saver activation, the virtual machine boots the MiG Linux image in which all grid jobs will be running. As soon as the screen saver deactivates, the virtual machine is shut down, and all resources are given back to the local user.

As shown in Figure 2, the virtual machine is just a normal process running in the host OS, which typically would be Windows. The virtual machine emulates the underlying physical hardware, thus creating a secure sandbox environment that allows an application written for one OS to be executed in another.

While the MiG sandboxes are quite similar to the ‘normal’ native resources in the MiG, a few things differ. Most importantly, since most typical desktop computers reside behind NAT routers it is not possible to achieve the MiG requirement of inbound ssh access on resourcers. Therefore, to stay in the philosophy of minimum intrusion, sandboxes use a strict pull-based model where all communication is initiated from the resource. For further details, see [5]. For MiG users who wish to deploy their applications on the virtualized sandbox platform, all that is necessary is to specify the ’sandbox’ keyword in the job description file.

2.4 WebCom

Problem solving for parallel systems traditionally lay in the realm of message passing systems such as PVM and MPI on networks of distributed machines, or in the use of specialised variants of programming languages like Fortran and C on distributed shared memory supercomputers. The WebCom System[8] relates more closely to message passing systems, although it is much more powerful. Message passing ar-

chitectures normally involve the deployment of a codebase on client machines, and employ a master or server to transmit or *push* messages to these clients.

Technologies such as PVM, MPI and other metacomputing systems place the onus on the developer to implement complete parallel solutions. Such solutions require a vast knowledge on the programmer's part in understanding the problem to be solved, decomposing it into its parallel and sequential constituents, choosing and becoming proficient in a suitable implementation platform, and finally implementing necessary fault tolerance and load balancing/scheduling strategies to successfully complete the parallel application. Even relatively trivial problems tend to give rise to monolithic solutions requiring the process to be repeated for each problem to be solved.

WebCom removes much of these traditional considerations from the application developer; allowing solutions to be developed independently of the physical constraints of the underlying hardware. It achieves this by employing a two level architecture: the computing platform and the development environment. The computing platform is implemented as an Abstract Machine (AM), capable of executing applications expressed as Condensed Graphs. Expressing applications as Condensed Graphs greatly simplifies the design and construction of solutions to parallel problems. The Abstract Machine executes tasks on behalf of the server and returns results over dedicated sockets. The computing platform is responsible for managing the network connections, uncovering and scheduling tasks, maintaining a balanced load across the system and handling faults gracefully. Applications developed with the development environment are executed by the abstract machine. The development environment used is specific for Condensed Graphs. Instructions are typically composed of both sequential programs (also called atomic instructions) and Condensed nodes encapsulating graphs of interacting sequential programs. In effect, a Condensed Graph on WebCom represents a hierarchical job control and specification language. The same Condensed Graphs programs execute without change on a range of implementation platforms from silicon based Field Programmable Gate Arrays[9] to the WebCom metacomputer and the Grid.

2.4.1 Architecture Overview

The WebCom abstract machine is constructed from a set of modules that plug into a module called the *backplane*. Each module in the WebCom system falls into one of two categories: it is either a *Core* module or a *User* module. Modules are loaded based on a initial configuration, thus bootstrapping the computational platform. Core modules include the *Compute Engine*, *Fault Tolerance*, Scheduling and Load balancing via the *Distributor*, *Security*, *Communications*, *Job Management* and *Information Management*. User modules can be provided to add additional functionality to the WebCom abstract machine. This architecture is outlined in Figure 3.

Compute Engine: The compute engine is responsible for low level task execution. Task composition varies depending on the compute engine in use. The default Con-

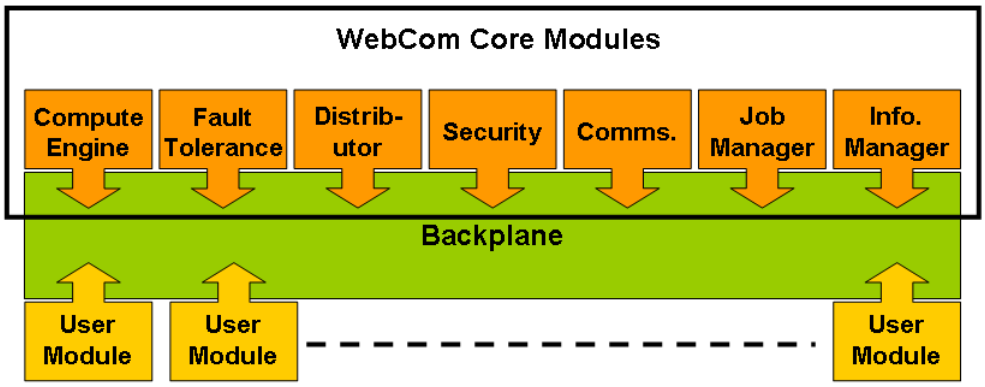


Figure 3: WebCom Abstract Machine Architecture showing how WebCom modules plug into the Backplane module.

densed Graphs compute engine is responsible for executing applications expressed as Condensed Graphs.

Fault Tolerance: The fault tolerance module detects and corrects faults that occur within the Abstract Machine. Mechanisms in place for fault tolerance range from simply rescheduling failed tasks to employing a unique processor replacement strategy [10].

Distributor: The distributor is responsible for allocating work to clients. The distributor operates according to a set of installed policies. Typically, a system-wide default policy exists that allows the distributor to select clients for task execution. These policies are plugable and hierarchal in nature and specify items such as the selection algorithm and associated configuration parameters. Nodes within a Condensed Graphs application are executed according to the installed policy. In addition, nodes themselves can specify their own distribution policy. Such nodes will be allocated to clients based on that policy. This provides great flexibility within the AM, as an application is not tied to one particular scheduling algorithm. Different nodes in a single application can be scheduled using different algorithms. It is possible that multiple applications executing on a single abstract machine can execute using multiple selection algorithms within the distributor.

Security: The security manager is responsible for authenticating the tasks, results and other messages transmitted throughout the WebCom infrastructure. The current security manager is based on the *Keynote*[11] standard.

Communications Manager Module: The communications manager is responsible for communicating tasks to clients. Once the distributor has decided where to allocate

a task, it is placed in a queue for the selected client. The communications manager module serves this queue, transmitting the task to the client. Tasks are sent based upon a *Pull Request* mechanism. When a client is willing to accept work, it issues a pull request. A WebCom will respond to this request by *pushing* the task to the client.

Job Manager: Each application within the WebCom AM executes within its own job space. The job manager [12] can be used to monitor the progress of job execution, pause and restart jobs and also suspend jobs.

2.5 Combined Potential

By utilising the strengths both the MiG and WebCom platforms, we have identified a number of areas where this collaboration can provide significant advances for both user communities. We will elaborate on some of these advantages in the remainder of this section.

For the MiG user community: MiG users submit through a script. Once the job enters the MiG system, the user has to periodically poll it to enquire about the jobs status. In most cases, the user will have a rough idea as to how often they need to issue this poll, however, it is possible that they may not issue this poll in a timely manner. Using WebCom as a submission engine for MiG, the user will be notified when the job has been completed. This is determined by WebCom issuing the poll for the job status at regular intervals. At worst, the maximum time the user will be waiting for such a notification would be equivalent to the polling interval the MiG Compute Engine employs.

A second potential benefit to the MiG user community is that this system provides a platform whereby users can construct scientific workflows using an intuitive interface. Just drag and drop the required nodes, connect the arcs and execute the resulting application. As the work presented here describes our initial investigations, it is apparent that a more complete suite of nodes would be required. However, this is easily achievable.

A third benefit to the MiG user community could be to support the PRC model of computing at the resource level. Resource donators can choose which projects their particular sandbox will support. With an ever increasing number of projects being supported by MiG, WebCom's targeting mechanism can be used to provide a more flexible approach to application execution, targeting applications to particular sandboxes. This would happen transparently to the user.

For the WebCom user community: The sandbox model may be of particular interest to users of WebCom. Creating jobs in a sandboxed environment allows for greater scope in deployment of WebCom. WebCom itself could be pre-packaged with a particular configuration. MiG could be used to deploy and instantiate any number

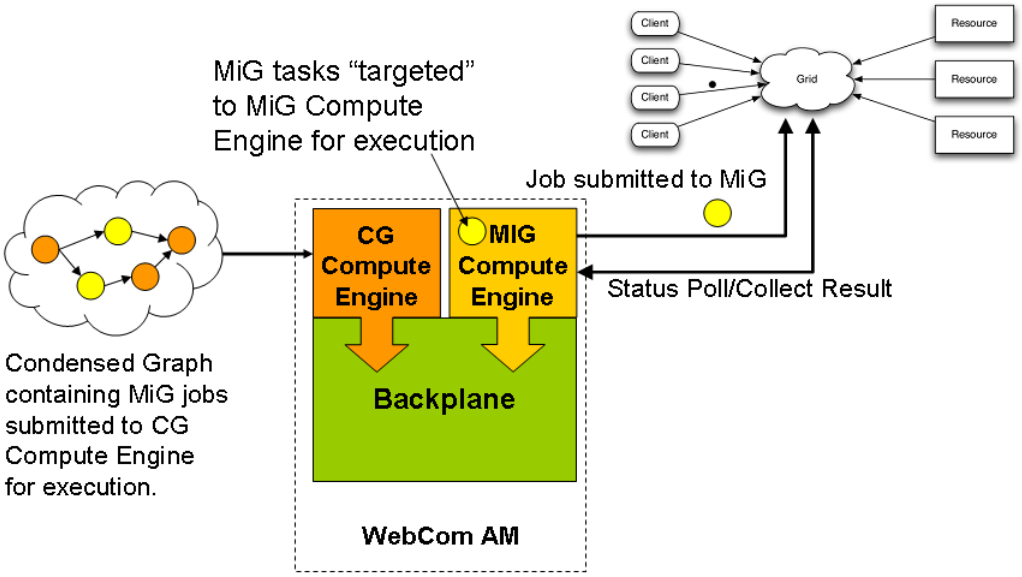


Figure 4: High Level computation showing MiG compute engine

of these packages. When executed, a dynamic WebCom platform would be created, allowing users to execute applications within a fresh, secure environment. This has potential for creating multiple WebCom instances whose lifetime may only exist for the duration of a single application.

WebCom's targeting can be use to include MiG jobs in a higher level workflow. Scientific computations are typically better suited to native execution. Native MiG computations could be easily invoked to leverage the underlying resources.

3 Implementation

At the basis of supporting MiG interaction from WebCom lies the client side of the MiG system shown on the left in Figure 1. When incorporating a MiG client in another system, the 'minimum intrusion' property of MiG is highly valuable, as the only requirements to the client is a valid X.509 certificate trusted by MiG and outbound HTTP and HTTPS. There are several ways for users to interact with the MiG system: Either through a standard web browser or by several types of scripts. A job is defined by a minimal Resource Specification Language(mRSL) text file. The user holding the certificate automatically gets a home directory and must upload all grid-related files to this home directory. Then, all file names mentioned in the job description are relative to the user's home directory, just as if it were local.

Initial support for MiG interpretability is provided at two levels: the first being the provision of a new Engine Module called the MigEngine and the second the provision of a suite of nodes that can be included in the WorkFlow being developed. At the

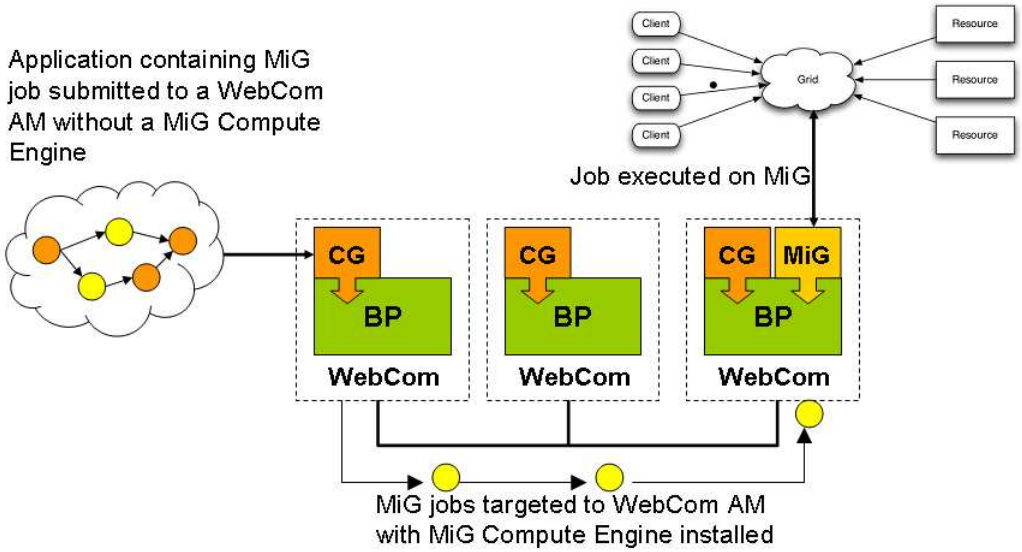


Figure 5: Multiple WebCom machines, showing targeting of MiG jobs.

simplest level a node that executes a MiG job description file is provided. When the MiG compute engine receives an appropriate instruction, it carries out the operations required for that instruction.

The instruction to execute a job submission script file goes through a number of stages: First the appropriate mRSL file is saved to some predetermined location, next the MiG compute engine invokes the MiG platforms submission script. The result of this submission is a job id. This id can be used to obtain information about the jobs progression such as its current status and what result was received. Typically, within the MiG system, these operations are conducted periodically by the user.

WebComs MiG Compute Engine can be used to remove this task from the user. When the compute engine submits the task for execution it then proceeds to poll the mig system at regular intervals to obtain the status of a job.

In the initial version of the MiG Compute engine, once a job finishes it returns the jobs status to the user. At this stage the user can manually retrieve any output files, error files generated by the job.

On an extended WebCom network, Figure 5, it is possible for a user to construct workflow applications containing MiG jobs without having to install or join the MiG platform. When such a workflow is submitted to a WebCom not capable of executing it directly (that WebCom does not have a MiG Compute Engine installed locally), it advertises the need for another WebCom that has a a MiG Compute Engine plugged in.

Other WebCom instances on the network that match the requested criteria will respond. When a response is received the task will subsequently be scheduled to a responder and WebComs targeting mechanism is then invoked to ensure the task is

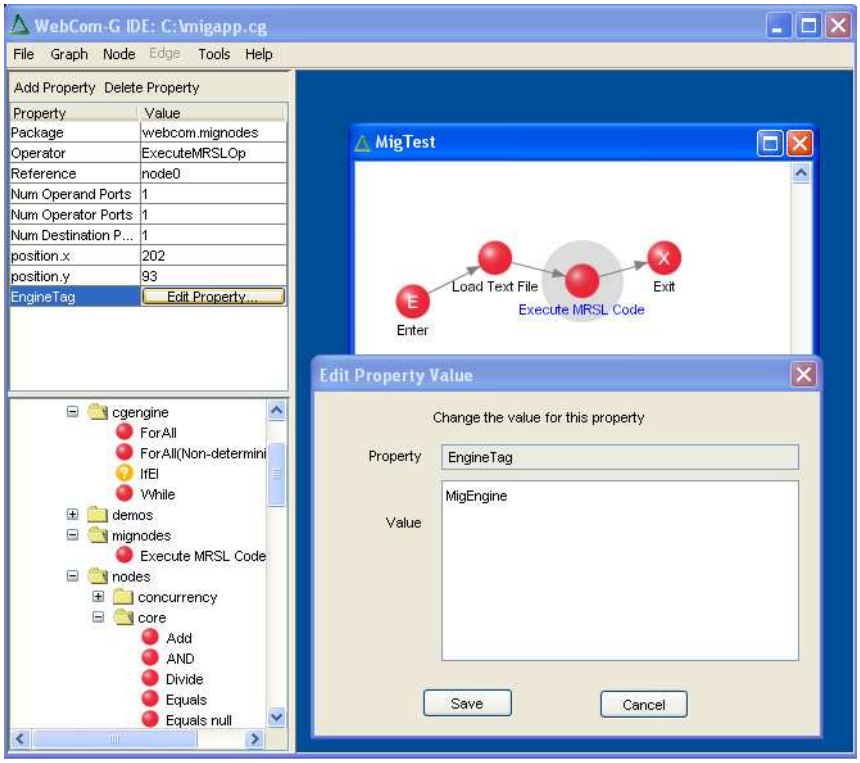


Figure 6: Simple Application developed in the WebCom IDE

delivered to the selected responder. In the case where there are multiple responders to a request, the first responder is the one that will be selected. In the case where there are no responders (no WebCom has a MiG Compute Engine plugged in), the task is put into a waiting queue. Periodically, tasks in this queue are passed to the distributor for re-allocation. This results in a further request for a WebCom with a MiG compute engine. Tasks will continue to wait, possibly indefinitely, until a suitable responder is found.

4 Test Application

To illustrate the process of executing a MiG application, a simple application was created. This is shown in Figure 6. The test application simply loads a mRSL file and feeds that as input to the `Execute MRSL Code` task. The final result is then passed back to the user. Typically nodes in a condensed graph will be targeted for execution on a Condensed Graphs compute engine. However, the `Execute MRSL Code` node shown in the figure must be targeted for execution on a MiG Compute Engine. This targeting information can be seen in the *Edit Property Value* dialog that is visible. This is necessary, because as described earlier it is the MiG Compute Engine that possesses the knowledge as how to execute this type of node. When the

`Execute MRSL Code` node finishes the status is passed on to the `X` node.

The test configuration for this application included one computer that acted as a MiG gateway: This computer had the MiG submission system and a WebCom instance installed. The WebCom system installed on this machine had both the Condensed Graphs and MiG execution engine modules installed.

A second machine with WebCom was used for constructing, and executing the application. Once the application was submitted, execution proceeded as expected: The specific MiG nodes were targeted to the WebCom instance with the MiG compute engine installed.

5 Conclusions & Future Work

This paper presented details of our initial investigation into supporting interactions between two different computing systems: The Minimum Intrusion Grid (MiG) and WebCom. Initial investigations concentrated on providing added functionality to the WebCom system that facilitate the targeting and execution of MiG jobs. Applications consisting of MiG jobs are created within the WebCom IDE and executed on the WebCom computational platform. The initial implementation proved successful although there is significant scope for further research.

Further research can be conducted in to the number of potential benefits to both systems user communities: such as more fine grained application targeting for MiG users, deploying and establishing dynamic WebCom compute environments, investigating the use of more scientific computations such as Fast Fourier Transformations for example and extending the palette of nodes available to the users when creating the workflows.

In addition to the items outlined above, further research and gathering of results generated by MiG can be conducted. The current prototype implementation just returns the status code to the user, along with the job id. Currently, the user has to manually retrieve the results.

Additional nodes can be provided for the complete suite of MiG commands, each command would then have a direct correspondent when creating visual applications.

Acknowledgements

The work presented here was part funded by the Boole Centre for Research in Informatics, Science Foundation Ireland under the WebCom-G project and the Danish NABIIT program committee.

References

[1] I. Foster, C. Kesselman: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, Elsevier Press (2004) 593-620

[2] I. Foster, C. Kesselman:The Globus project: a status report. Proceedings of the Seventh Heterogeneous Computing Workshop, March 1998, IEEE Computer Society Press, pp. 4–19

[3] B. Vinter: The Architecture of the Minimum intrusion Grid: MiG. Proceedings of Communicating Process Architectures 2005

[4] J.P. Morrison, B. Clayton, D. Power, A. Patil: WebCom-G: Grid Enabled Metacomputing- The Journal of Neural, Parallel and Scientific Computation. Special Issue on Grid Computing. Editors H.R. Arabnia, G.A. Gravvanis, M.P. Bekakos, Vol. 12(3), PP. 419-438, September, 2004

[5] R. Andersen, B. Vinter: Harvesting Idle Windows CPU Cycles for Grid Computing. Proceedings of GCA-2006

[6] D.P. Anderson: BOINC: a system for public-resource computing and storage. Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)

[7] R. Figueiredo, P. Dinda, J. Fortes: A Case for Grid Computing on Virtual Machines. Proceedings of the International Conference on Distributed Computing Systems (ICDCS), May 2003

[8] J.P. Morrison, D.A. Power, J.J. Kennedy: An Evolution of the WebCom Metacomputer. The Journal of Mathematical Modelling and Algorithms: Special issue on Computational Science and Applications, 2003(2), pp 263-276

[9] J.P. Morrison, P.J. O'Dowd. P.D. Healy: Searching RC5 Keyspaces with Distributed Reconfigurable Hardware. Proceedings of ERSA 2003, Las Vegas, June 23-26, 2003

[10] J.J. Kennedy: Design and Implementation N-Tier Metacomputer with Decentralised Fault Toerance. PhD Thesis, University College Cork, Ireland, May 2004

[11] M. Blaze et al: The Keynote Trust Management System, Version 2. Internet Request for Comments 2704. September 1999

[12] N. Cafferkey, P.D. Healy, D.A. Power, J.P. Morrison: Job Management in WebCom. In Proceedings of the 6th International Symposium on Parallel and Distributed Computing (ISPDC) - IEEE Press, Hagenberg, Austria, July 4th-8th, 2007

Appendix D

Publication 4

Proceedings of the 2008 International Conference on Grid Computing & Applications, GCA 2008, Las Vegas, Nevada, USA. CSREA Press 2008, ISBN 1-60132-068-X, pp. 175-181

Rasmus Andersen, Brian Vinter: The Scientific Byte Code Virtual Machine

The Scientific Byte Code Virtual Machine

Rasmus Andersen
University of Copenhagen
eScience Centre
2100 Copenhagen, Denmark
Email: rasmus@diku.dk

Brian Vinter
University of Copenhagen
eScience Centre
2100 Copenhagen, Denmark
Email: vinter@diku.dk

***Abstract*—Virtual machines constitute an appealing technology for Grid Computing and have proved a promising mechanism that greatly simplifies and enforces the employment of grid computer resources.**

While existing sandbox technologies to some extent provide secure execution environments for applications deployed in a heterogeneous platform as the Grid, they suffer from a number of problems including performance drawbacks and specific hardware requirements.

This project introduces a virtual machine capable of executing platform-independent byte codes specifically designed for scientific applications. Native libraries for the most prevalent applications domains mitigate the performance penalty. As such, grid users can view this machine as a basic grid computing element and thereby abstract away the diversity of the underlying real compute elements.

Regarding security, which is of great concern to resource owners, important aspects include stack isolation by using a harvard memory architecture, and no support for neither I/O nor system calls to the host system.

Keywords: Grid Computing, virtual machines, scientific applications.

I. Introduction

Although virtualization was first introduced several decades ago, the concept is now more popular than ever and has revived in a multitude of computer system aspects that benefit from properties such as platform independence and increased security. One of those applications is grid Computing[5] which seeks to combine and utilize distributed, heterogeneous resources as one big virtual supercomputer. Regarding utilization of the public's computer resources for grid computing, virtualization, in the sense of virtual machines, is a necessity for fully leveraging the true potential of grid computing. Without virtual machines, experience shows that people are, with good reason, reluctant to put their resources on a grid where they have to not only install and manage a software code base, but also allow native execution of unknown and untrusted programs. All these issues can be eliminated by introducing virtual machines.

As mentioned, virtualization is by no means a new concept. Many virtual machines exist and many of them have been combined with grid computing. However, most of these were designed for other purposes and suffer from a few problems when it comes to running high performance scientific applications on a heterogeneous computing platform. Grid computing is tightly bonded to eScience, and while standard jobs may run perfectly and satisfactory in existing virtual machines, 'gridified' eScience jobs are better suited for a dedicated virtual machine in terms of performance.

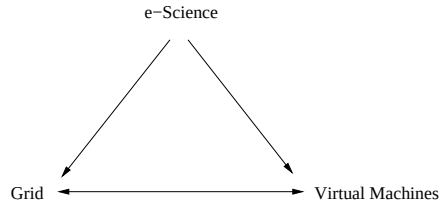


Fig. 1. Relatationship between VMs, the Grid, and eScience

Hence, our approach addresses these problems by developing a portable virtual machine specifically designed for scientific applications: The Scientific Byte Code Virtual Machine (SciBy VM).

The machine implements a virtual CPU capable of executing platform independent byte codes corresponding to a very large instruction set. An important feature to achieve performance is the use of optimized native libraries for the most prevalent algorithms in scientific applications. Security is obviously very important for resource owners. To this end, virtualization provides the necessary isolation from the host system, and several aspects that have made other virtual machines vulnerable have been left out. For instance, the SciBy VM supports neither system calls nor I/O.

The following section (II) motivates the usage of virtual machines in a grid computing context and why they are beneficial for scientific applications. Next, we describe the architecture of the SciBy VM in Section III, the compiler in Section IV, related work in Section VI and conclusions in Section VII.

II. Motivation

The main building blocks in this project arise from properties from virtual machines, eScience, and a grid environment in a combined effort, as shown in figure 1.

The individual interactions impose several effects from the viewpoint of each end, described next.

A. eScience in a Grid Computing Context

eScience, modelling computationally intensive scientific problems using distributed computer networks, has driven the development of grid technology and as the simulations get

more and more accurate, the amount of data and needed compute power increase equivalently. Many research projects have already made the transition to grid platforms to accommodate the immense requirements for data and computational processing. Using this technology, researchers gain access to many networked computers at the cost of a highly heterogeneous computing platform. Obviously, maintaining application versions for each resource type is tedious and troublesome, and results in a deploy-port-redeploy cycle. Further, different hardware and software setups on computational resources complicate the application development drastically. One never knows to which resource a job is submitted in a grid, and while it is possible to assist each job with a detailed list of hardware and software requirements, researchers are better left off with a virtual workspace environment that abstracts a real execution environment.

Hence, a virtual execution environment spanning the heterogeneous resource platform is essential in order to fully leverage the grid potential. From the view of applications, this would render a resource access uniform and thus the much easier ”compile once run anywhere” strategy; researchers can write their applications, compile them for the virtual machine and have them executed anywhere in the Grid.

B. Virtual Machines in a Grid Computing Context

Due to the renewed popularity of virtualization over the last few years, virtual machines are being developed for numerous purposes and therefore exist in many designs, each of them in many variants with individual characteristics. Despite the variety of designs, the underlying technology encompasses a number of properties beneficial for Grid Computing [4]:

1) Platform Independence: In a grid context, where it is inherently intrinsic to move around application code as freely as application data, it is highly profitable to enable applications to be executed anywhere in the grid. Virtual machines bridge the architectural boundaries of computational elements in a grid by raising the level of abstraction of a computer system, thus providing a uniform way for applications to interact with the system. Given a common virtual workspace environment, grid users are provided with a compile-once-run-anywhere solution.

Furthermore, a running virtual machine is not tied to a specific physical resource; it can be suspended, migrated to another resource and resumed from where it was suspended.

2) Host Security: To fully leverage the computational power of a grid platform, security is just as important as application portability. Today, most grid systems enforce security by means of user and resource authentication, a secure communication channel between them, and authorization in various forms. However, once access and authorization is granted, securing the host system from the application is left to the operating system.

Ideally, rather than handling the problems after system damage has occurred, harmful - intentional or not - grid applications should not be able to compromise a grid resource in the first place.

Virtual machines provide stronger security mechanisms than conventional operating systems, in that a malicious process running in an instance of a virtual machine is only capable of destroying the environment in which it runs, i.e. the virtual machine.

3) Application Security: Conversely to disallowing host system damage, other processes, local or running in other virtualized environments, should not be able to compromise the integrity of the processes in the virtual machine.

System resources, for instance the CPU and memory, of a virtual machine are always mapped to underlying physical resources by the virtualization software. The real resources are then multiplexed between any number of virtualized systems, giving the impression to each of the systems that they have exclusive access to a dedicated physical resource. Thus, grid jobs running in a virtual machine are isolated from other grid jobs running simultaneously in other virtual machines on the same host as well as possible local users of the resources.

4) Resource Management and Control: Virtual machines enable increased flexibility for resource management and control in terms of resource usage and site administration. First of all, the middleware code necessary for interacting with the Grid can be incorporated in the virtual machine, thus relieving the resource owner from installing and managing the grid software. Secondly, usage of physical resources like memory, disk, and CPU usage of a process is easily controlled with a virtual machine.

5) Performance: As a virtual machine architecture interposes a software layer between the traditional hardware and software layers, in which a possibly different instruction set is implemented and translated to the underlying native instruction set, performance is typically lost during the translation phase. Despite of recent advances in new virtualization and translation techniques, and the introduction of hardware-assisted capabilities, virtual machines usually introduce performance overhead and the goal remains achieving near-native performance only. The impact depends on system characteristics and the applications intended to run in the machine.

To summarize, virtual machines are an appealing technology for Grid Computing because they solve the conflict between the grid users at the one end of the system and resource providers at the other end. Grid users want exclusive access to as many resources as possible, as much control as possible, secure execution of their applications, and they want to use certain software and hardware setups.

At the other end, introducing virtual machines on resources enables resource owners to service several users at once, to isolate each application execution from other users of the system and from the host system itself, to provide a uniform execution environment, and managed code is easily incorporated in the virtual machine.

C. A Scientific Virtual Machine for Grid Computing

Virtualization can occur at many levels of a computer system and take numerous forms. Generally, as shown in Figure 2, virtual machines are divided in two main categories:

System virtual machines and process virtual machines, each branched in finer division based on whether the host and guest instruction sets are the same or different. Virtual machines with the same instruction set as the hardware they virtualize do exist in multiple grid projects as mentioned in Section VI. However, since full cross-platform portability is of major importance, we only consider *emulating* virtual machines, i.e. machines that execute another instruction set than the one executed by the underlying hardware.

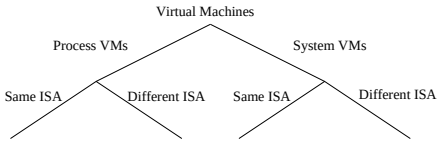


Fig. 2. Virtual machine taxonomy. Similar to Figure 13 in [9]

System virtual machines allow a host hardware platform to support multiple complete guest operating systems, all controlled by a virtual machine monitor and thus acting as a layer between the hardware and the operating systems. Process virtual machines operate at a higher level in that they virtualize a given platform for user applications. A detailed description of virtual machines can be found in [9].

The overall problem with system virtual machines that emulate the hardware for an entire system, including applications as well as an operating system, is the performance loss incurred by converting all guest system operations to equivalent host system operations, and the implementation complexity in developing a machine for every platform type, each capable of emulating an entire hardware environment for essentially all types of software.

Since the application domain in focus is scientific applications only, there is really no need for full-featured operating systems. As shown in Figure 3, process level virtual machines are simpler because they only execute individual processes, each interfaced to the hardware resources through a virtual instruction set and an Application Binary Interface.

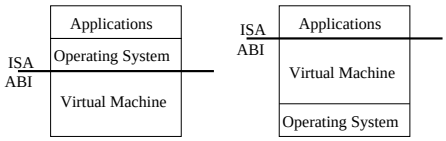


Fig. 3. System VMs (left) and Process VMs (right)

Using the process level virtual machine approach, the virtual machine is designed in accordance with a software development framework. Developing a virtual machine for which there is no corresponding underlying real machine may sound counterintuitive, but this approach has proved successful in several cases, best demonstrated by the power and usefulness of the Java Virtual Machine. Tailored to the Java programming language, it has provided a platform independent computing environment for many application domains, yet there is no commonly used real Java machine [1].

Similar to Java, applications for the SciBy VM are compiled into a platform independent byte code which can be executed on any device equipped with the virtual machine. However, applications are not tied to a specific programming language. As noted earlier, researchers should not be forced to rewrite their applications in order to use the virtual machine. Hence, we produce a compiler based on a standard ansi C compiler.

D. Enabling Limitations

While the outlined work at hand may seem comprehensive, especially the implementation burden with virtual machines for different architectures, there are a some important limitations that greatly simplify the project. Firstly, the implementation burden is lessened drastically by only giving support for running a single sequential application. Giving support for entire operating systems is much more complex in that it must support multiple users in a multi-process environment, and hardware resources such as networking, I/O, the graphics processor, and 'multimedia' components of currently used standard CPUs are also typically virtualized.

Secondly, a virtual machine allows fine-grained control over the actions taken by the code running in the machine. As mentioned in Section VI, many projects use sandbox mechanisms in which they by various means check all system instructions. The much simpler approach taken in this project is to simply disallow system calls. The rationale for this decision is that:

- scientific applications perform basic calculations only
- using a remote file access library, only files from the grid can be accessed
- all other kinds of I/O are not necessary for scientific applications and thus prohibited
- indispensable systems calls must be routed to the grid

III. Architectural Overview

The SciBy Virtual Machine is an abstract machine executing platform independent byte codes on a virtual CPU, either by translation to native machine code or by interpretation. However, in many aspects it is designed similarly to conventional architectures; it includes an Application Binary Interface, an Instruction Set Architecture, and is able to manipulate memory components. The only thing missing in defining the architecture is the hardware. As the VM is supposed to be run on a variety of grid resources, it must be designed to be as portable as possible, thereby supporting many different physical hardware architectures.

Based on the previous sections, the SciBy VM is designed to have 3 fundamental properties:

- Security
- Portability
- Performance

That said, all architectural decisions presented in the following sections rely solely on providing portability. Security is

[1]The Java VM has been implemented in hardware in the Sun PicoJava chips

obtained by isolation through virtualization, and performance is solely obtained by the use of optimized native libraries for the intended applications and taking advantage of the fact that scientific applications spend most of their time in these libraries. The byte code is as such not designed for performance. Therefore, the architectural decisions do not necessarily seek to minimize code density, minimize code size, reduce memory traffic, increase the average number of clock cycles per instruction, or other architectural evaluation measurements, but more for simplicity and portability.

A. Application Binary Interface

The SciBy VM ABI defines how compiled applications interface with the virtual machine, thus enabling platform independent byte codes to be executed without modification on the virtual CPU.

At the lowest level, the architecture defines the following machine types arranged in big endian order:

- 8-bit byte
- 16-, 32-, or 64-bit halfword
- 32-, 64-, or 128-bit word
- 64-, 128-, or 256-bit doubleword

In order to support many different architectures, the machine exists in multiple variations with different word sizes. Currently, most desktop computers are either 32- or 64-bit architectures, and it probably won't be long before we see desktop computers with 128-bit architectures. By letting the word size be user-defined, we capture most existing and near-future computers.

Fundamental primitive data types include, all in signed two's complement representation:

- 8-bit character
- integers (1 word)
- single-precision floating point (1 word)
- double-precision floating point (2 words)
- pointer (1 word)

The machine contains a register file of 16384 registers, all 1 word long. This number only serves as a value for having a potentially unlimited amount of registers. The reasons for this are twofold. First of all due to forward compatibility, since the virtual register usage has to be translated to native register usage, in which one cannot tell the upper limit on register numbers. So basically, in a virtual CPU, one should be sure to have more registers than the host system CPU. Currently, 16384 registers should be more than enough, but new architectures tend to have more and more registers. Secondly, for the intended applications, the authors believe that a register-based architecture will outperform a stack-based one[8]. Generally, registers have proved more successful than other types of internal storage and virtually every architecture designed in the last few decades uses a register architecture.

Register computers exist in 3 classes depending on where ALU instructions can access their operands, register-register architectures, register-memory architectures and memory-memory architectures. The majority of the computers shipped nowadays implement one of those classes in a 2- or 3-operand format. In order to capture as many computers as possible, the SciBy VM supports all of these variants in a 3-operand format, thereby including 2-operand format architectures in that the destination address is the same as one of the sources.

B. Instruction Set Architecture

One key element that separates the SciBy VM from conventional machines is the memory model: The machine defines a Harvard memory architecture with separate memory banks for data and instructions. The majority of conventional modern computers use a von Neumann architecture with a single memory segment for both instructions and data. These machines are generally more vulnerable to the well-known buffer overflow exploits and similar exploits derived from 'illegal' pointer arithmetic to executable memory segments. Furthermore, the machine will support hardware setups that have separate memory pathways, thus enabling simultaneous data and instruction fetches. All instructions are fetched from the instruction memory bank which is inaccessible for applications: All memory accesses from applications are directed to the data segment. The data memory segment is partitioned in a global memory section, a heap section for dynamically allocated structures, and a stack for storing local variables and function parameters.

1) Instruction Format: The instruction format is based on *byte codes* to simplify the instruction stream. The format is as follows: Each instruction starts with a one-byte *operation code* (opcode) followed by possibly more opcodes and ends with zero or more operands, see Figure 4. In this sense, the machine is a multi-opcode multi-address machine. Having only a single one-byte opcode limits the instruction set to only 256 different instructions, whereas multiple opcodes allows for nested instructions, thus increasing the number of instructions exponentially. A multi-address design is chosen to support more types of hardware.

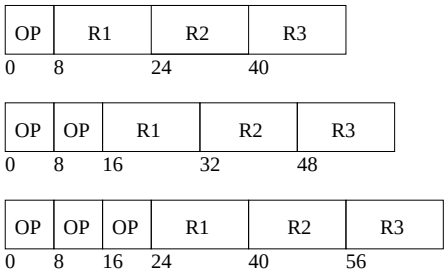


Fig. 4. Examples of various instruction formats on register operands.

2) Addressing Modes: Based on the popularity of addressing modes found in recent computers, we have selected 4 addressing modes for the SciBy VM, all listed below.

- Immediate addressing: The operand is an immediate, for instance MOV R1 4 which moves the number 4 to register 1.
- Displacement addressing: The operand is an offset and a register pointing to a base address, for instance ADD

R1 R1 4(R2) which adds to R1 the value found 4 words from the address pointed out by R2.

- Register addressing: Operand is a register, for instance MOV R1 R2
- Register indirect addressing: Address part is a register containing the address of an operand, for instance ADD R1, R1, (R2), which adds to R1 the value found at the address pointed out by R2.

3) Instruction Types: Since the machine defines a Harvard architecture, it is important to note that data movement is carried out by `LOAD` and `STORE` operations which operate on words in the data memory bank. `PUSH` and `POP` operations are available for accessing the stack.

Table I summarizes the most basic instructions available in the SciBy VM. Almost all operations are simple 3-address operations with operands, and they are chosen to be simple enough to be directly matched by native hardware operations.

Instruction group	Mnemonic
Moves	load, store
Stack	push, pop
Arithmetic	add, sub, mul, div, mod
Boolean	and, or, xor, not
Bitwise	and, or, shl, shr, ror, rol
Compare	tst, cmp
Control	halt, nop, jmp, jsr, ret, br, be_eq, br_lt, etc

TABLE I
BASIC INSTRUCTION SET OF THE SCIBY VM

While these instructions are found in virtually every computer, they exist in many different variations using various addressing modes for each operand. To accommodate this and assist the compiler as much as possible, the SciBy VM provides regularity by making the instruction set orthogonal on both operations, data types, and the addressing modes. For instance the 'add' operation exists in all 16 combinations of the 4 addressing modes on the two source registers for both integers and floating points. Thus, the encoding of an 'add' instruction on two immediate source operands takes up 1 byte for choosing arithmetic, 1 byte to select the 'add' on two immediates, 2 bytes to address one of the 16384 registers as destination register and then 16 bytes for each of the immediates, yielding a total instruction length of 36 bytes.

C. Libraries

In addition to the basic instruction set, the machine implements a number of basic libraries for standard operations like floating-point arithmetic and string manipulation. These are extensions to the virtual machine and are provided on an per-architecture basis as statically linked native libraries optimized for specific hardware.

As explained above, virtual machines introduce a performance overhead in the translation phase from virtual machine object code to the native hardware instructions of the underlying real machine. The all-important observation here is that scientific applications spend most of their running time executing 'scientific instructions' such as string operations, linear algebra, fast fourier transformations, or other library functions. Hence, by providing optimized native libraries, we can take advantage of the synergy between algorithms, the compiler translating them, and the hardware executing them.

Equipping the machine with native libraries for the most prevalent scientific algorithms and enabling future support for new libraries increases the number of potential instructions drastically. To address this problem, multiple opcodes allows for nested instructions as shown in Figure 5. The basic instructions are accessible using only one opcode, whereas a floating point operation is accessed using two opcodes, i.e. *FP_lib FP_sub R1 R2 R3*, and finally, if one wishes to use the `WFTA` instruction from the `FFT_2` library, 3 opcodes are necessary: *FFT_lib FFT_2 WFTA args*.

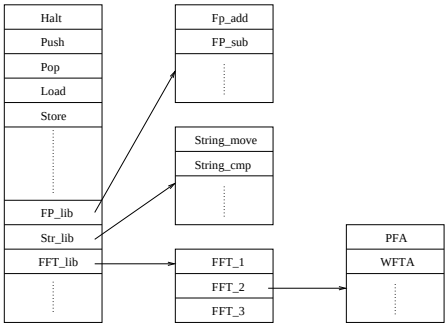


Fig. 5. Native libraries as extension to the instruction set

A special library is provided to enable file access. While most grid middlewares use a staging strategy that downloads all input files prior to the job execution and uploads output files afterwards, the MiG-RFA [1] library accesses files directly on the file server on an on-demand basis. Using this strategy, an application can start immediately, and only the needed fragments of the files it accesses are transferred.

Ending the discussion of the architecture, it is important to re-emphasize that all focus in this part of the machine is on portability. For instance, when evaluating the architecture, one might find that:

- Having a 3-operand instruction format may give unnecessarily large code size in some circumstances
- Studies may show that the displacement addressing mode is typically used to nearby addresses, thereby suggesting that these instructions only need a few bits for the operand
- Using register-register instructions may give unnecessarily high instruction count in some circumstances
- Using byte codes increases the code density
- Variable instruction encoding decreases performance

Designing an architecture includes a lot of trade-offs, and even though many of these issues are zeroed by the interpreter or translator, the proposed byte code is far from optimal by normal architecture metrics. However, the key point is that we target only a special type of applications on a very broad hardware platform.

IV. Compilation and Translation

While researchers do not need to rewrite their scientific applications for the SciBy VM, they do need to compile their application using a SciBy VM compiler that can translate the high level language code to the SciBy VM code. While developing a new compiler from scratch of course is a possibility, it is also a significant amount of work which may prove unprofitable since many compilers designed to be retargetable for new architectures already exist.

Generally, retargetable compilers are constructed using the same classical modular structure: A front end that parses the source file, and builds an intermediate representation, typically in the shape of a parse tree, used for machine-independent optimizations, and a back end that translates this parse tree to assembly code of the target machine.

When choosing between open source retargetable compilers, the set of possibilities quickly narrows down to only a few candidates: GCC and LCC. Despite the pros of being the most popular and widely used compiler with many supported source languages in the front end, GCC was primarily designed for 32-bit architectures, which greatly complicates the retargeting effort. LCC however, is a light-weight compiler, specifically designed to be easily retargetable to a new architecture.

Once compiled, a byte code file containing assembly instruction mnemonics is ready for execution in the virtual machine, either by interpretation or by translation, where instructions are mapped to the instruction set of the host machine using either a load-time or run-time translation strategy. Results remain to be seen, yet the authors believe that in case a translator is preferable to an interpreter, the best solution would be load-time translator, based on observations from scientific applications:

- their total running time is fairly long which means that the load-time penalty is easily amortized
- they contain a large number of tight loops where run-time translation is guaranteed to be inferior to load-time translation

V. Experiments

To test the proposed ideas, a prototype of the virtual machine has been developed, in the first stage as a simple interpreter implemented in C. There is no compiler yet, so all sample programs are hand-written in assembly code with the only goal of giving preliminary results that will show whether development of the complete machine can be justified.

The first test is a typical example of the scientific applications the machine targets: A Fast Fourier Transform (FFT). The program first computes 10 transforms on a vector of varying sizes, then checksums the transformed vector to verify the result. In order to test the performance of the virtual machine, the program is also implemented in C to get the native base line performance, and in Java to compare the results of the SciBy VM with an existing widely used virtual machine.

The C and SciBy VM programs make use of the fftw library[6], while the Java version uses an FFT algorithm from the SciMark suite[7]. Obviously, this test is highly unfair in disfavor of the Java version for several reasons. Firstly, the fftw library is well-known to give the best performance, and comparing hand-coded assembly with compiler-generated high-level language performance is a common pitfall. However, even though Java-wrappers for the fftw library exist, it is essential to put these comparisons in a grid context. If the grid resources were to run the scientific applications in Java Virtual Machine, the programmers - the grid users - would not be able to take advantage of the native libraries, since allowing external library calls breaks the security of the JVM. Thereby, the isolation level between the executing grid job and the host system is lost[2]. In the proposed virtual machine, these libraries are an integrated part of the machine, and using them is perfectly safe.

Vector size	Native	SciBy VM	Java
524288	1.535	1.483	7.444
1048576	3.284	3.273	19.174
2097152	6.561	6.656	41.757
4194304	14.249	14.398	93.960
8388608	29.209	29.309	204.589

TABLE II
Comparison of the performance of an FFT application on a 1.86 GHz Intel Pentium M processor, 2MB cache, 512 MB RAM

Vector size	Native	SciBy VM	Java
524288	0.879	0.874	4.867
1048576	1.857	1.884	10.739
2097152	3.307	3.253	23.520
4194304	6.318	6.354	50.751
8388608	13.045	12.837	110.323

TABLE III
Comparison of the performance of an FFT application on a dual core 2.2 GHz AMD Athlon 4200 64-bit, 512 kB cache per core, 4GB RAM

As shown in Table II the FFT application is run on the 3 machines using different vector size, $2^{19}, ..., 2^{23}$. The results show that the SciBy VM is on-par with native execution, and that the Java version is clearly outperformed.

Since the fftw library is multithreaded, we repeat the experiment on a dual core machine and on a quad dual-core machine. The results are shown in Table III and Table IV.

From these results it is clear that for this application there

[2]In fact there is a US Patent (#6862683) on a method to protect native libraries

Vector size	Native	SciBy VM	Java
524288	0.650	0.640	4.955
1048576	1.106	1.118	12.099
2097152	1.917	1.944	27.878
4194304	3.989	3.963	61.423
8388608	7.796	7.799	134.399

TABLE IV
Comparison of the performance of an FFT application on a quad dual-core Intel Xeon CPU, 1.60 GHz, 4MB cache per core, 8GB RAM

Appendix E

Publication 5

Submitted to the Journal of Grid Computing, 2009. **Rasmus Andersen, Brian Vinter: Performance and Portability of the SciBy VM**

Journal of Grid Computing manuscript No.
(will be inserted by the editor)

Performance and Portability of the SciBy Virtual Machine

Rasmus Andersen · Brian Vinter

Received: date / Accepted: date

Abstract The Scientific Bytecode Virtual Machine is a virtual machine designed specifically for performance, security, and portability of scientific applications deployed in a Grid environment. The performance overhead normally incurred by virtual machines is mitigated using native optimized scientific libraries, security is obtained by sandboxing techniques. Lastly, by executing platform-independent bytecodes, the machine is highly portable.

To evaluate the machine, we demonstrate several use-case scenarios from some of the intended application domains. Further, we show the ease of porting the machine and distributing its jobs to a variety of predominant architectures and compare the results with native execution.

Keywords Grid Computing · Volunteer Computing · Virtual Machine · Scientific Applications

1 Introduction

In this project, we try to combine strengths from Volunteer Computing and Grid Computing [11] to form a very powerful computing platform. To harness the compute power from this platform, and put in on tap for researchers with immediate needs for massive amounts of CPU-cycles, we have introduced the Scientific Bytecode Virtual Machine (SciBy VM) [5].

Designed specifically for scientific applications, we take advantage of some well-known facts about scientific applications to achieve close to optimal performance on a variety of hardware platforms. Secondly, by executing a platform-independent bytecode, the machine is higly portable, analogously to the Java Virtual Machine [17]. While

R. Andersen
E-mail: rasmus@diku.dk

B. Vinter
eScience Centre, University of Copenhagen, Universitetsparken 1, 2100 Copenhagen
Tel.: +45 353-21421
E-mail: vinter@diku.dk

JVM is not appropriate for high performance computing, it has proved extremely successful for a number of application domains due to the ease of portability. Finally, as security cannot be underestimated when allowing arbitrary code to execute on donated resources, the machine implements several layers of security to ensure not only the host machine, but also the integrity of the application.

2 Motivation

As a computing platform, Volunteer Computing has shown its potential with many successful projects utilizing the idle time from the donated resources. For instance, at the time of writing, the very popular Folding@Home project [9] operates at whapping 4.3 petaflop/s. In comparison, the Roadrunner supercomputer tops the supercomputer top 500 list with a peak performance of 1.1 petaflop/s.

A natural extension to the Volunteer Computing scheme is to 'gridify' it. In their current form, primarily based on the BOINC framework [6], these public resource computing projects are merely one-way systems; one can only donate resources. A resource has a pre-installed program that continuously fetches new *data* to execute, as opposed to a Grid, where resources continuously fetch new *programs*. The entry cost for a researcher to actually submit new programs to these Volunteer Computing systems is high enough to render them useless for smaller research projects in need of compute power [1].

In our perception, an important aspect of Grid Computing is to allow seamless access for researchers to the huge amount of connected resources, thus creating the illusion of unified access to one big supercomputer. Gaining widespread acceptance as an important computing platform depends on the ability of making the technology accessible to general exploitation, not only for computer specialists, but also for researchers from other areas, since they are the real end-users of the systems.

The need for compute power increases rapidly, primarily driven by eScience, i.e. computationally intensive scientific simulations with enormous amounts of data. As the simulations get more and more accurate, the requirements to the compute platform increase equivalently.

'Gridifying' a Volunteer Computing system, i.e. not only making it easy for the public to donate their idle resources, but also enabling researchers to utilize these public resources, gives a whole new dimension to a Grid. However, the two systems differ significantly on two aspects: Security and portability. Firstly, a resource in a Volunteer Computing system only hosts one single application that keeps getting new data, while a resource in Grid system hosts arbitrary untrusted applications. Hence, there is a substantial difference in the security level to ensure the host system. Secondly, while a 'standard' Grid is typically composed of Linux/Unix desktop computers, clusters, and supercomputers from different trusted compute centres, a volunteer computing grid drastically increases the heterogeneity of the system. For instance, the client statistics from the Folding@Home project reveals that the biggest contributors are NVIDIA GPUs and PlayStation 3's, markedly trailing ATI GPUs and Windows machines. Linux and Mac machines are barely noticeable. Obviously, porting a single Volunteer Com-

[1] According to the BOINC website, `http://boinc.berkeley.edu/trac/wiki/BoincIntro`, it takes 3 man-months to port an existing application to the BOINC framework and about $5000 for hardware used for project management

puting application to these architectures is considerably easier than enabling arbitrary programs to run on any architecture.

With the abundance of new powerful architectures in personal computers and devices, we now face many radically different CPU architectures, GPU architectures, operating systems, software environments, and specific limitations to memory, network, and disk usage. Therefore, enabling execution of arbitrary code on connected resources requires a lot of work in specifying an execution environment. Ultimately, before non-computer specialists can adopt this computing platform, it is necessary to hide these complexities of the Grid system, and free the applications from all architectural details of the computational resources connected to the Grid.

Specifying or standardizing an execution environment - an executable code format and runtime environment - and ensuring the safety of the host machine, is by no means an easy task. However, as outlined in [5], virtual machines can help bridge the architectural boundaries by raising the abstraction level from the underlying hardware, thus enabling moving around application code as freely as application data. The Java Virtual Machine Specification [13] and the Common Intermediate Language [15] from MicroSoft are two examples that have proved very useful with many different virtual machine implementations. Once compiled into the byte code of the respective standards, an application can run on any of these machines. The drawback of these systems is the requirement of using a specialized programming language rarely used in scientific applications, and the performance drawback from executing byte codes instead of native machine code.

The approach taken in this project is to use a hybrid model, in which a completely platform-independent - and thus highly portable - bytecode is augmented with the ability of calling native libraries to improve performance. With respect to security, virtualization reduces the problem of requiring donators to trust arbitrary programs to only trust the virtual machine that executes the programs. In effect, virtualization increases the application integrity and eases the resource management and control.

3 Related Work

During the last decade, virtual machines [18] have been revived and used for many different purposes. VMware [27], Xen [20], and VirtualBox [24] are some of the most popular *system* virtual machines. With the ability of hosting multiple complete guest operating systems, applications can be isolated safely, and recent progress - even with hardware support - has provided near-native speed. However, they are typically tied to a limited set of platform architectures and difficult to port.

Application virtual machines, for instance JVM and .Net VM, only support running a single application. Applications are expressed as portable intermediate language representations [13, 15] and executed by abstract virtual machines [17, 19]. As opposed to system virtual machines, these machines are highly portable, yet their performance drawback hinders them from being used for high performance computing.

Virtual machines are just one way of providing a `sandbox`, i.e. a confined environment in which a host system allows untrusted 3rd-party code to execute without compromising the host. Native code sandboxes allows a native application to execute under a well-defined set of constraints, all expressed in native code themselves. Since the introduction by Wahbe [23], the literature contains many variants for different architectures. The vx32 [10] would be of primary interest for a Volunteer System since it

has minimal intrusion on the host system, thus lowering the workload for the people trying to donate their resources. Where many similar systems require modifications to the host system, for instance kernel modifications, special privileges and permissions, vx32 runs on unmodified host systems. Applications just need to be linked to a user-level library, which then sandboxes the application in a secure execution environment. vx32 is limited to x86 architectures.

An upcoming interesting sandbox is Google's Native Client [26]. Aimed at browser-based applications, the system tries to run compute-intensive applications in the address space of the browser at native speed. When an application is sent to a browser, a Native Client plugin loads the Native Client container, which is a sandbox containing native libraries enabling the application to execute at native speed. Like vx32, Native Client is limited to x86 architectures, and since the workload required to express all types of security restrictions in native machine code is quite substantial, porting to other architectures is unlikely to happen within foreseeable future.

4 The SciBy Virtual Machine

In the Sciby VM, we combine intermediate bytecode in a virtual machine with native libraries to achieve near-native performance on many different architectures. To our knowledge, this is the first research project that introduces this type of hybrid machine.

From a user perspective, the SciBy Virtual Machine [5] is an abstract virtual machine capable of executing platform-independent bytecodes corresponding to a single sequential application. Users will need to use a designated compiler to translate their source code into the bytecode, which is then sent to and executed by an interpreter on any available resource. The compiler is based on `gcc` [2], thus taking benefit of the many source languages accepted by the front end of the compiler; only the back end has been modified to target the SciBy architecture.

For deployment as a virtual machine for scientific applications in a merged Grid and Volunteer Computing system, the machine is designed solely for portability, host system security, and performance.

4.1 Security

Host system security is ensured by a virtual machine to isolate untrusted code in a sandbox. Typically, this type of software based fault isolation [23] focuses on disallowing unsafe instructions access to memory outside the sandbox, illegal instructions, privileged instructions, etc. In SciBy VM, we made the deliberate choice of disallowing system calls, including all types of I/O, altogether. Thus, we only allow instructions that perform transformations on data, control flow instructions, and data movement instructions. Dedicated for scientific applications, there is really no need for system calls, and the only type of I/O necessary, is access to input files and output files; this is achieved using the Remote File Access library, presented in Section 4.4. Indispensable system calls will be routed back to the Grid for execution.

A typical sandbox feature, which the SciBy VM also implements, is a Harvard memory model with separate segments for data and code to ensure correct access to data, and preventing jumps to instructions outside the code area. Using this model, the

machine is less vulnerable to typical exploits derived from 'illegal' pointer arithmetic to other executable memory segments.

4.2 Portability

Portability is obtained by designing a completely platform-independent bytecode and by virtualizing a very broad hardware platform. The instruction set includes all typical instructions for data transformation, control flow, and data movement. To capture most of the physical computer platforms, it is designed as an orthogonal multi-opcode multi-address set of instructions in a 3-operand format, thereby permitting easy translation to 2-operand architectures. Addressing modes include immediate addressing, displacement addressing, register addressing, and indirect addressing.

Virtualization occurs at many levels in various computer subsystems. It typically provides an illusion of hardware configurations that are not physically available, for instance virtual memory which gives each process the illusion of having exclusive access to the entire address space of the machine. With the SciBy VM, much focus is placed on being forward compatible by virtualizing hardware setups of the future. Most notably, as the tendency goes towards more and more registers, the machine provides a virtually unlimited number of registers. Just to provide a number for the compiler, it is set to 16384 128-bit registers. Further, as there is now a shift from 32 bit towards 64 bit architectures, the next step probably being 128 bit, the SciBy VM supports all these word sizes.

4.3 Performance

Performance is obtained by augmenting the instruction categories with one essential category: Instructions that can call external native library functions. Executing bytecodes in a virtual machine will incur performance overhead, but the key point here is to utilize the fact that scientific applications spend most of their time in these libraries. The characteristics of these applications, for instance bioinformatics, high-energy physics, or image processing analysis, is a small but very time-consuming code size. Typically, these application set up input and output files, and then enter some dense loop structures in which time-consuming calls to an external library are continuously made. In the SciBy VM, the surrounding code is replaced by the portable bytecode, but with support for calling native optimized libraries in order to achieve near native performance, this is illustrated in Figure 1.

Obviously, native libraries are not portable and a version of the machine must be equipped with statically linked libraries for every architecture, and made available for people willing to participate. However, with the success of a library follows ports to other architectures, and it is a simple and small task to embed a library in the machine.

4.4 Remote File Access

As mentioned above, the SciBy VM does not have access to a file system on the host machine. Instead of transferring input and output files to the resource executing a job, the files remain in the home directory of the job owner at a file server, and are then

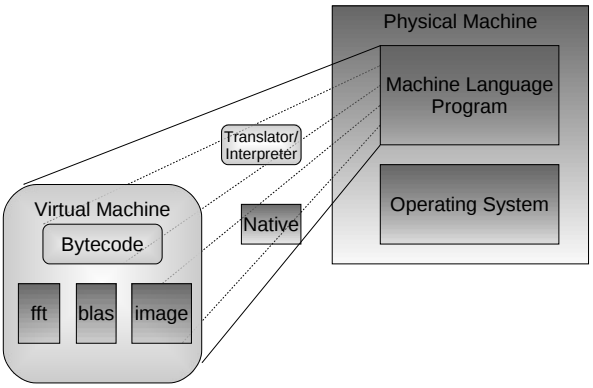


Fig. 1 An untrusted application converted to bytecode, running in VM with embedded scientific native libraries.

accessed remotely using the Remote File Access Library [4] (RFA) in the Minimum intrusion Grid, described below. Imposed as a user-level library between an application and the host operating system, it intercepts all file access routines and directs them to the server on which the file in question resides.

This type of on-demand remote file access enables the resource to limit its file retrieval to a set of file fragments holding the needed data, thus only downloading needed data and only uploading modified data. Especially for scientific applications, this technique can be a big advantage compared to the traditional staging technique. On initialization, a job can start right away; it does not need to wait for potentially huge files to be transferred from a file server. And often, only the first parts, or scattered fragments of a file are really needed. For instance, high-energy physics applications typically find their data in huge database input files, but the amount of blocks read from the file is relatively small. Similarly, output files are written remotely on demand.

Using prefetching to hide the latency, and a strategy for dynamically adjusting block sizes in the data transfer that adapts to the application's data access pattern, and mitigates fluctuating network conditions, the file access library has proved to have very good performance. Compared to staging techniques, tests have shown that speedups of above one thousand are well within the realm of possibilities. In optimal scenarios, it can even be superior to local file access, for instance with CPU-bound applications, where the block transfer times, and more importantly, the system calls for the file access, can be completely hidden by computations.

5 Deployment

While both the SciBy virtual machine and the remote file access library were independently developed and applicable in many systems, they are targeted at, and deployed in, the Minimum intrusion Grid.

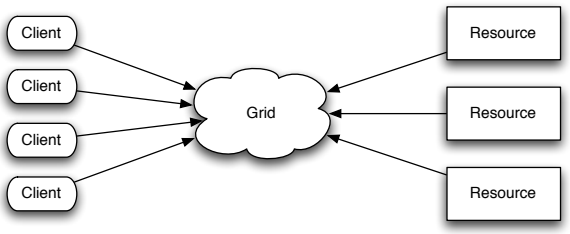


Fig. 2 The abstract MiG model

5.1 Minimum intrusion Grid

MiG [22, 16] is a stand-alone approach to Grid that does not depend on any existing systems, i.e. it is a completely separate platform for Grid computing. The philosophy behind the MiG is to provide a Grid infrastructure that imposes as few requirements on users and resources as possible.

Driven by the aim of minimum intrusion on participating parties, the idea is to ensure that users only need a signed X.509 certificate, trusted by Grid, and a web browser that supports HTTP and HTTPS. A resource only needs to create an MiG user on the system and to support inbound ssh and outbound HTTPS. Initially, the resource must register to the MiG system using a certificate.

By keeping the Grid system disjoint from both users and resources, as shown in Figure 2, this model allows the Grid system to appear as a centralized black-box to both users and resources, and all upgrades and trouble shooting can be performed locally within the Grid without intervention from neither users nor resource administrators. Thus, all functionality is placed in a physical Grid system.

The basic functionality in MiG starts with users submitting jobs to MiG and resources sending requests for jobs. A resource then receives an appropriate job from MiG, executes the job, and sends the result to MiG that can inform the user of the job completion. Thus, MiG provides full anonymity; users and resources interact only with MiG, never with each other.

5.2 Porting SciBy VM and Distributing jobs

The process of porting the SciBy VM to other architectures is straightforward. The interpreter is written in strict ANSI-C making it highly portable. In all experiments presented in the next section, the source files were copied to the test machine and compiled. The applications were compiled into bytecode files on a single machine and then transferred for immediate execution. As regards the libraries, they were all available for the tested platforms. So once built on the particular platform, the library is embedded into the virtual machine for the platform in question.

People willing to donate can then download a version of the machine matching their platform from the website, and just like the native code sandbox [3] provided by the MiG, screen saver software will contact the Grid for a job on activation. During download, a resource description file is generated stating resource features such as which

libraries are available. It is then up to the scheduler to match the resource description with job description files, as shown below.

Researchers in need of compute power compile their source code into the SciBy VM bytecode, for instance *fourier.bin*, which needs to be uploaded to one's home directory in MiG. Next, submitting the job to the MiG can be done either through the website using an X.509 certificate or by downloading scripts to automate the process. The job description, in MiG known as a mRSL file, used for submitting the job to the MiG is:

```
::EXECUTE::
scibyvm fourier.bin
::NOTIFY::
jabber: my_id@jabber.org
::INPUTFILES::
fourier.bin
::OUTPUTFILES::
outputimage.pgm
::CPUTIME::
900
::ARCHITECTURE::
SciBy-VM
::RUNTIMEENVIRONMENT::
libfourier
```

The MiG has already run the native code sandbox volunteer computing project, based on existing virtual machines, for a long time, and has proved flexible enough to deal with resources behind NAT routers, and with such dynamic and transient compute elements as screen saver resources.

6 Evaluation

The Fast Fast Fourier Transform (FFT) is an obvious choice for evaluating the SciBy VM, since it is a fundamental kernel in so many scientific applications, for instance data compression, fluid dynamics, seismic imaging, image processing, computer tomography, data filtering, spectral analysis, and digital signal processing. The core of these applications is the necessity of computing Fourier transforms, and the performance of this type of application relies heavily on the routines available for performing the transforms.

In this test, our application uses the `fftw` [12] library to perform 10 transforms on a vector of varying sizes, $2^{19}, ..., 2^{23}$, and then checksums the result vector to verify the result. The application is compared to a native C-version on 3 different machines:

- A 1.86 GHz Intel Pentium M, 2 MB cache, 512 MB RAM
- A dual core 2.2 GHz AMD Athlon 4200 64-bit, 512 kB cache per core, 4 GB RAM
- A dual quad core Intel Xeon, 1.60 GHz, 4 MB cache per core, 8 GB RAM

As shown in Table 1, the performance of the SciBy VM is on par with native execution, and when taking advantage of a multi-threaded library, it can utilize multi-core architectures.

To give a deeper evaluation of the performance and portability, we invent new experiments as examples of typical scientific applications and evaluate them on different

	Pentium M		AMD Athlon		Intel Xeon	
Vector size	Native	SciBy VM	Native	SciBy VM	Native	Sciby VM
524288	1.535	1.483	0.879	0.874	0.650	0.640
1048576	3.284	3.273	1.857	1.884	1.106	1.118
2097152	6.561	6.656	3.307	3.253	1.917	1.944
4194304	14.249	14.398	6.318	6.354	3.989	3.963
8388608	29.209	29.309	13.045	12.837	7.796	7.799

Table 1 Comparison of SciBy VM and native performance on an FFT application on 3 different architectures.

architectures. The results in all experiments are the average of 3 consecutive runs, all measured in seconds using the Linux `time` command. Firstly, we turn to the field of image processing, then, we perform some experiments using a `BLAS` library.

6.1 Image Processing

Image processing is widely used in many scientific applications, such as medical imaging, computer graphics rendering, sensing and detection systems, and in general, over the last few years it is becoming a topic of interest for a broad scientific community. Using `fourier` [8], a portable image processing and analysis library, we write a sample application that applies a series of transformations, for instance `gaussian adaptive smoothing` filter, on a raw 2592x1944 pixel `pgm` image. We then run the application in 4 different setups:

- Native: Run natively, with the image in the local file system
- SciBy VM: Run inside the SciBy VM with the image in the local file system (suspending the local file access restriction)
- Native + RFA: Run natively, using the Remote File Access library to access the file 200 km away (standard 5 Mbps ADSL-line).
- SciBy VM + RFA: Run inside the SciBy VM using the RFA library as above.

These tests are performed on 4 architectures:

- An Intel Core 2 1.86 GHz processor, 2GB memory, running 32 bit Ubuntu linux,
- An AMD Athlon 64-bit processor 3000+, 1GB memory, running Debian-amd64
- A 3.2 GHz PowerPC 64-bit processor, 2 hardware threads, from the Cell Broadband Engine, running 32-bit Yellow Dog Linux[2]
- an Intel Core 2 2.4 GHz processor, 4GB memory, running Mac OSX.

Table 2 shows that there is no overhead when running inside the virtual machine in any of the setups. Since the entire file is used in an unbalanced fashion, there is no gain from using the RFA library. The time to transfer the image using `curl` and `lighttpd` was 46.293 seconds, which is exactly the difference between the runs with and without RFA. Thus, a staging technique would be equal to accessing the file remotely.

Next, to illustrate utilization of a multi-core architecture, we use the `diplib` [21] image processing library, which is multi-threaded. In this test, we apply the very compute-intensive second order derivative Laplace filter on the image, and execute on the 8-core Intel Xeon 1.60 GHz with 8 GB memory. Using the `taskset` command, the experiment is carried out using 1,2,4, and 8 cores.

[2] The `fourier` image library does not utilize the SPEs in the Cell BE.

Table 2 Results from the fourier application on a single-core host machine including transfer time of the image data from Grid

Arch.	Native	SciBy	Native+RFA	SciBy+RFA
Core 2	96.36	96.41	96.87	96.90
AMD	78.93	78.89	78.75	79.01
PPC	166.18	164.25	167.76	166.21
Mac	58.72	59.05	59.55	60.01

Table 3 Results from the `diplib` application on an 8-core host machine. The image transfer time, 46 seconds, is not included

Cores	Native	SciBy	Native+RFA	SciBy+RFA
1	101.53	101.34	147.47	147.80
2	51.36	51.94	97.76	97.60
4	27.74	27.63	73.23	73.54
8	15.14	15.23	61.87	61.90

Table 4 Results from the video processing, including transfer time of the image from the Grid

Arch.	Native	SciBy	Native+RFA	SciBy+RFA
Core 2	376.58	377.11	261.81	262.01
AMD	260.24	260.13	192.89	193.11

From the results in Table 3, we can once again conclude that there is no overhead from using the virtual machine. And, there is immediate support for multi-core utilization. Again, since the `diplib` image reads the entire image before it processes it, there is no gain from the remote file access library.

To wrap up image processing performance tests, a final example uses the `ffmpeg` [7] library to decode frames from a video file. Each decoded frame is then transformed using the `Imlib2` [14] image library. In this case, we just a apply a simple blurring effect to every frame. Since the `ffmpeg` balances I/O and processing, i.e. it consecutively reads data for a single frame, and then decodes it, the RFA library can take advantage of the prefetching, thus having every frame available in the instant it is needed. Therefore, as shown in 4, using the RFA library is faster than using local file access. The test is performed on the Intel and AMD computers only.

6.2 Basic Linear Algebra Subroutines

`BLAS` (Basic Linear Algebra Subroutines)is widely used for high-performance computing and benchmarking. `BLAS` is a set of efficient routines for most of the basic vector and matrix operations. They are widely used as the basis for other high quality linear algebra software packages, for instance `lapack` and `linpack`. In this test, we perform a series of operations from the ATLAS [25] and GotoBLAS [1] libraries on a 500 by 500 matrix. It is all computed in memory, so there is no file access. The results displayed in 5 once again show that the SciBy VM achieves near native speed.

Table 5 Results from the BLAS benchmark

Arch.	Native	SciBy VM
Core 2	38.466	38.211
AMD	46.340	46.870
PPC	43.513	43.662
Mac	*result not available yet*	

7 Conclusions and Future Work

Merging Volunteer Computing and Grid Computing systems is a big step towards realizing one of the initial ideas of Grid Computing, namely harnessing idle CPU power from all types of computer resources and putting them on tap for world-wide sharing. To harness the compute power from an ever-increasing farm of architectures and use it for scientific research, 3 fundamental obstacles have been identified and dealt with: portability, security, and performance. To this end, we have presented and evaluated the SciBy Virtual Machine.

Security is an all-important topic when utilizing other people's computers. The virtual machine ensures host system integrity by isolating the application in a secure execution environment.

Bytecodes and a virtual machine executing them provides platform-independence at the cost of performance. In this paper we have introduced a hybrid model, where the machine executes mobile bytecodes and uses native optimized libraries to mitigate the performance drawback. Thus, the model applies best to applications that spend most of their time in the libraries, which is the case for scientific applications.

Using typical scientific libraries, we have evaluated the machine's performance and found it to be on par with native execution. For other types of applications, the machine will currently perform poorly, but lessons learned from similar machines will surely alleviate this problem.

References

1. *Gotoblas*, `http://www.tacc.utexas.edu/resources/software/`.
2. *The gnu compiler collection*, `http://gcc.gnu.org`.
3. Rasmus Andersen and Brian Vinter, *Harvesting idle windows cpu cycles for grid computing*, GCA (Hamid R. Arabnia, ed.), CSREA Press, 2006, pp. 121–126.
4. _____, *Direct application access to grid storage: Research articles*, Concurr. Comput. : Pract. Exper. **19** (2007), no. 9, 1287–1298.
5. _____, *The scientific byte code virtual machine*, GCA, 2008, pp. ppp–ppp.
6. David P. Anderson, *Boinc: A system for public-resource computing and storage*, GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (Washington, DC, USA), IEEE Computer Society, 2004, pp. 4–10.
7. Fabrice Bellard, *ffmpeg*, `http://www.ffmpeg.org/`.
8. M. Emre Celebi, *fourier*, `http://mac.softpedia.com/get/Development/Libraries/Fourier.shtml`.
9. Folding@home, *Folding@home distributed computing*, `http://folding.stanford.edu/`.
10. Bryan Ford and Russ Cox, *Vx32: lightweight user-level sandboxing on the x86*, ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference (Berkeley, CA, USA), USENIX Association, 2008, pp. 293–306.
11. Ian Foster, *The grid: A new infrastructure for 21st century science*, Physics Today **55(2)** (2002), 42–47.

12. Matteo Frigo and Steven G. Johnson, *The design and implementation of FFTW3*, Proceedings of the IEEE **93** (2005), no. 2, 216–231, special issue on "Program Generation, Optimization, and Platform Adaptation".
13. James Gosling, Bill Joy, Guy Steele, and Gilad Bracha, *Java language specification*, third ed., Addison-Wesley Professional, July 2005.
14. Carsten Haitzler, *Imlib2*, `http://docs.enlightenment.org/api/imlib2/html/`.
15. ECMA International, *Standard ecma-335 - common language infrastructure (cli)*, 4 ed., June 2006.
16. Henrik Hoey Karlsen and Brian Vinter, *Minimum intrusion grid - the simple model*, WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (Washington, DC, USA), IEEE Computer Society, 2005, pp. 305–310.
17. Tim Lindholm and Frank Yellin, *The java(tm) virtual machine specification (2nd edition)*, Prentice Hall PTR, April 1999.
18. William Mcewan, *Virtual machine technologies and their application*, Accessed 14 March 2003 www.ddj.com/documents/s=882/ddj0008f/ 0008f.htm, 2002, pp. 55–62.
19. Erik Meijer, Redmond Wa, and John Gough, *Microsoft clr overview*.
20. Ian Pratt, Keir Fraser, Steven Hand, Christian Limpach, Andrew Warfield, Dan Magenheimer, Jun Nakajima, and Asit Mallick, *Xen 3.0 and the art of virtualization*, Proceedings of Linux Symposium 2005, July 2005.
21. Michael van Ginkel, *diplib*, `http://www.diplib.org`.
22. Brian Vinter, *The Architecture of the Minimum intrusion Grid (MiG)*, Communicating Process Architectures 2005, sep 2005, pp. –.
23. Robert Wahbe, Steven Lucco, Thomas E. Anderson, and Susan L. Graham, *Efficient software-based fault isolation*, In Proceedings of the 14th ACM Symposium on Operating Systems Principles, 1993, pp. 203–216.
24. Jon Watson, *Virtualbox: bits and bytes masquerading as machines*, Linux J. **2008** (2008), no. 166, 1.
25. R. Clint Whaley and Antoine Petitet, *Minimizing development and maintenance costs in supporting persistently optimized BLAS*, Software: Practice and Experience **35** (2005), no. 2, 101–121.
26. Bennet Yee and David Sehr, *Native client: A sandbox for portable, untrusted x86 native code*, Tech. report, 2001.
27. *Vmware*, `http://www.vmware.com/`.

Appendix F

Publication 6

Recent Developments in Grid Technology and Applications. G.A. Gravvanis, J.P. Morrison, H.R. Arabnia and D.A. Power, Editors **Rasmus Andersen, Martin Rehr, Brian Vinter: Cycle-Scavenging in Grid Computing**

Chapter 1

CYCLE-SCAVENGING IN GRID COMPUTING

Rasmus Andersen, Martin Rehr, and Brian Vinter*
University of Copenhagen, Department of Computer Science

PACS 05.45-a, 52.35.Mw, 96.50.Fm. **Keywords:** .

Key Words: cycle scavenging, Grid Computing, Public Resource Computing, Minimum intrusion Grid, Virtual Machines **A**MS Subject Classification: 53D, 37C, 65P.

Abstract

Grid Computing and Public Resource Computing systems each provide means of obtaining and utilizing distributed computational resources. In this chapter we explore the benefits and potential of combining the two fields into a system that offers the flexibility of Grid Computing and the resource richness of the cycle-scavenging PRC systems.

Important aspects of a combined effort include host system security, application security, platform independence, and resource management and control.

Sandboxing technology accommodates these requirements, and two different models offered by the Minimum intrusion Grid are presented. Both of them use virtual machines to sandbox Grid Computing jobs on idle personal desktop computers and have proved to close the gap between Grid Computing and Public Resource Computing.

*E-mail address: vinter@diku.dk

1 Introduction

Cycle scavenging, or Screen Saver Science, is an increasingly popular computing paradigm used within many fields of science that seek to tap the enormous amount of unused processing power from the millions of computers connected to the Internet. The paradigm is best known from the many successful Public Resource Computing, PRC, projects, such as SETI@Home, where the idle time cycles are used for a dedicated scientific application.

Grid Computing [6] and PRC each provide increasingly interesting means of obtaining computational resources. Grid Computing is mostly used for connecting university supercomputers, while PRC is predominantly used by research projects for harvesting PC-based idle CPU cycles for a small number of research projects. However, even though the two fields appear closely related, little effort has been made to combine them into a system that offers the flexibility of Grid computing with the resource richness of the PRC model.

One of the Grid promises is to make it possible to share and effectively use distributed resources on an unprecedented scale. Specifically, this includes harnessing the unused capacity of idle desktop PCs. Harnessing 'free' cycles through PRC is of great interest since a modern PC is powerful and highly underutilized, and as such cycle harvesting provides huge calculation potential if one combines millions of them in a computing Grid. A lot of research has been done to Grid-enable idle resources, yet no widely accepted system to effectively scavenge idle cycles, in particular idle Windows cycles, has been found. Most known Grid systems such as ARC which is based on the Globus toolkit and Condor [3] are unsuitable for PRC computing, as they work under the underlying assumption that the resources are available indefinitely, while PRC resources are transient by nature.

Ensuring the safety of a donated resource while it executes a Grid job in a PRC context is an all important topic since all free resources will vanish if the model proves harmful to the hosts. Contrary to standard PRC tasks, a Grid job may take any form and include the execution of any binary. Therefore it is necessary to take precautions to ensure that the execution of Grid jobs cannot harm donated resources neither intentionally nor by accident.

Extending the PRC concept to actual Grid Computing, security and installation of software on the host resource are equally important issues to a successful solution. All PRC projects known to the authors require the donor to install software on the resource that should contribute, which alone eliminates users from donating resources from computers that they do not have administrative rights on. The software installation also opens up possible exploits and requires the donor to perform updates on that software. This is not desirable and may reduce the amount of donated resources.

Our goal is to design Grid PRC sandbox models that with a minimal effort from the resource owner can execute Grid jobs in a secure environment. This chapter addresses the problems that need to be solved in order to combine PRC and Grid Computing by scavenging idle desktops for general scientific use. First of all, under the restrictions mentioned above, a method to gain access to the CPU cycles on the idle resource must be found. In this aspect, security is a major issue. Ideally, a resource owner should neither install any software nor execute any foreign applications that, intentionally or not, could compromise his system. Secondly, the resource, possibly hidden by a Network Address Translation router and a firewall, must be attached to the Grid without forcing modification to routers or firewalls. Thirdly, it must be ensured that a given resource has installed the correct soft-

ware base that a given Grid job requires. Finally, we introduce extra features to improve the model; for instance, a method to predict the idle time period of a resource in advance. Using this method, a job with a time deadline is submitted to a resource that is predicted to be available in the specified time frame.

The approach taken here is to use virtual machines to provide a sandbox that by default includes everything needed to execute Grid jobs and completely separates them from the resource host system. This separation ensures that, on the one hand, a grid job cannot compromise the host system, and on the other hand, the grid job is protected from other users of the system. Two different models have been implemented: Firstly the The MiG-SSS, which is based on system virtual machines that execute an entire linux guest operating system specifically designed as a Grid resource. Secondly, the One-Click MiG Applet Resource, which is based on Java applet technology and enables users with a standard Java-enabled web browser to donate their idle time PCs with a single click on a website, thereby starting a Java applet capable of executing Grid jobs written for this framework.

1.1 Related Work

BOINC [2] is a software platform that allows many different distributed computing projects to utilize idle volunteered computer resources. Many Public Resource Computing systems use BOINC, and research groups can with little effort create new projects. A project involves a set of applications that will be run in a BOINC client on a user's resource. As such, BOINC could be used for this project by running the proposed virtual machine as the project application.

A few projects have been found to combine Screen Saver Science with Grid computing, for instance the Entropia Virtual Machine [4], which is a commercial product, and the survey [5] that is merely an extensive introduction to the approach of using virtual machines for Grid computing.

2 Sandboxing in a Grid Context

Although virtualization was introduced several decades ago, the concept is now more popular than ever and has revived in a multitude of computer system aspects that benefit from properties such as platform independence and increased security. One of those applications is Grid computing, where the ultimate goal of combining and utilizing distributed, heterogeneous resources as one big virtual supercomputer necessitates these properties. Regarding utilization of the public's computer resources for grid computing, virtualization, in the sense of virtual machines, is a necessity for fully leveraging the true potential of Grid computing. Without virtual machines, experience shows that people are, with good reason, reluctant to put their resources on a grid since they have to not only install and manage a software code base, but also allow native execution of unknown and untrusted programs. All these issues can be eliminated by introducing virtual machines.

eScience, modelling scientific problems using computers, has driven the development of Grid technology, and as the simulations get more and more accurate, the amount of data and needed compute power increase equivalently. Many research projects have already made the transition to Grid platforms to accommodate the immense requirements for data

and computational processing. Using this technology, researchers gain access to many networked computers at the cost of a highly heterogeneous computing platform. Obviously, maintaining application versions for each resource type is tedious and troublesome, and results in a deploy-port-redeploy cycle. Further, different hardware and software setups on computational resources complicate the application development drastically. One never knows to which resource a job is submitted in a grid, and while it is possible to assist each job with a detailed list of hardware and software requirements, researchers are better left off with a virtual workspace environment that abstracts a real execution environment.

Hence, a virtual execution environment spanning the heterogeneous resource platform is essential in order to fully leverage the grid potential. From the view of applications, this would render a resource access uniform and thus allow the much easier ”compile once run anywhere” strategy; researchers can write their applications, compile them for the virtual machine and have them executed anywhere in the Grid.

Due to the renewed popularity of virtualization over the last few years, virtual machines are being developed for numerous purposes and therefore exist in many designs, each of them in many variants with individual characteristics. Despite the variety of designs, the underlying technology encompasses a number of properties beneficial for Grid Computing:

Platform Independence: In a grid context, where it is intrinsic to move around application code as freely as application data, it is highly profitable to enable applications to be executed anywhere in the grid. Virtual machines bridge the architectural boundaries of computational elements in a grid by raising the level of abstraction of a computer system, thus providing a uniform way for applications to interact with the system. Given a common virtual workspace environment, grid users are provided with a compile-once-run-anywhere solution.

Furthermore, a running virtual machine is not tied to a specific physical resource; it can be suspended, migrated to another resource and resumed from where it was suspended.

Host Security: To fully leverage the computational power of a grid platform, security is just as important as application portability. Today, most grid systems enforce security by means of user and resource authentication, a secure communication channel between them, and authorization in various forms. However, once access and authorization is granted, securing the host system from the application is left to the operating system.

Ideally, rather than handling the problems after system damage has occurred, harmful - intentional or not - grid applications should not be able to compromise a grid resource in the first place.

Virtual machines provide stronger security mechanisms than conventional operating systems, in that a malicious process running in an instance of a virtual machine is only capable of destroying the environment in which it runs, i.e. the virtual machine.

Application Security: Conversely to disallowing host system damage, other processes, local or running in other virtualized environments, should not be able to compromise the integrity of the processes in the virtual machine.

System resources, for instance the CPU and memory, of a virtual machine are always mapped to underlying physical resources by the virtualization software. The real resources

are then multiplexed between any number of virtualized systems, giving the impression to each of the systems that they have exclusive access to a dedicated physical resource. Thus, grid jobs running in a virtual machine are isolated from other simultaneous grid jobs running in other virtual machines on the same host as well as possible local users of the resources.

Resource Management and Control: Virtual machines enable increased flexibility for resource management and control in terms of resource usage and site administration. First of all, the middleware code necessary for interacting with the Grid can be incorporated in the virtual machine, thus relieving the resource owner from installing and managing the grid software. Secondly, usage of physical resources like memory, disk, and CPU usage of a process is easily controlled with a virtual machine.

Performance: As a virtual machine architecture interposes a software layer between the traditional hardware and software layers, in which a possibly different instruction set is implemented and translated to the underlying native instruction set, performance is typically lost during the translation phase. Despite of recent advances in new virtualization and translation techniques, and the introduction of hardware-assisted capabilities, virtual machines usually introduce performance overhead and the goal remains achieving near-native performance only. The impact depends on system characteristics and the applications intended to run in the machine.

To summarize, virtual machines are an appealing technology when combining Grid Computing and PRC because they solve the conflict between the grid users at the one end of the system and resource providers at the other end. Grid users want exclusive access to as many resources as possible, as much control as possible, secure execution of their applications, and they want to use certain software and hardware setups. At the other end, introducing virtual machines on resources enables resource owners to service several users at once, to isolate each application execution from other users of the system and from the host system itself, to provide a uniform execution environment, and managed code is easily incorporated in the virtual machine.

3 Enabling the sandboxes for the Grid

The main problem with scavenging personal computers is that the vast majority are hidden behind a NAT router, i.e. they do not have global IP address and are therefore not reachable from the Internet. Hence, to enable the sandbox for Grid Computing, care must be taken to circumvent the missing inbound Internet access. Naturally, this issue is highly dependent on the Grid middleware in question. In this work, the sandboxes are designed for the Minimum intrusion Grid, MiG, which is presented next, before the details of how to tailor the two sandboxes for MiG are explained.

3.1 Minimum intrusion Grid

MiG [8] [7] is a stand-alone approach to Grid that does not depend on any existing systems, i.e. it is a completely new platform for Grid computing. The philosophy behind the MiG is

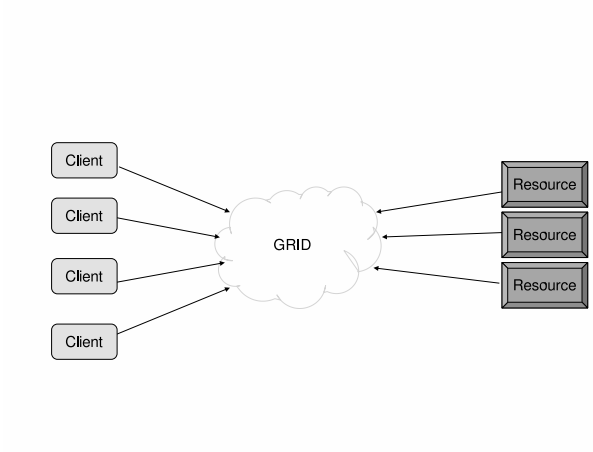


Figure 1: The abstract MiG model

to provide a Grid infrastructure that imposes as few requirements on users and resources as possible.

The idea is to ensure that users only need a signed X.509 certificate, trusted by Grid, and a web browser that supports HTTP and HTTPS. A resource only needs to create an MiG user on the system and to support inbound ssh and outbound HTTPS. Initially, the resource must register to the MiG system using a certificate.

By keeping the Grid system disjoint from both users and resources, as shown in Figure 1, this model allows the Grid system to appear as a centralized black-box to both users and resources, and all upgrades and trouble shooting can be performed locally within the Grid without intervention from neither users nor resource administrators. Thus, all functionality is placed in a physical Grid system that, although it appears as centralized system, in reality is a distributed system itself.

The basic functionality in MiG starts with users submitting jobs to MiG and resources sending requests for jobs. A resource then receives an appropriate job from MiG, executes the job, and sends the result to MiG that can inform the user of the job completion. Thus, MiG provides full anonymity; users and resources interact only with MiG, never with each other.

3.1.1 Scheduling

The centralized black box design of MiG makes it capable of strong scheduling, which implies full control of the jobs being executed and the resource executing them. Each job has an upper execution time limit, and when the execution time exceeds this time limit the job is rescheduled to another resource. This makes the MiG system very well suited to host PRC resources, as they by nature are very dynamic and frequently join and leave the Grid without notifying the Grid middleware.

3.2 Cycle-Scavenging using System Virtual Machines

In MiG-SSS, the screen saver model based on system virtual machine, the basic idea is to let resource owners install a sandbox to provide a secure execution environment in which the Grid job is completely isolated from the host machine and vice versa. Such a sandbox can take the shape of a virtual machine, which is exactly the approach that we have taken in this work. Two techniques can be used to provide a virtual machine: Emulation and Virtualization[5]. Emulation provides the functionality of the target processor completely in software, which makes it a very secure approach. Also, the ability to emulate one processor type on any other processor type makes it ideal for this scenario. However, the method of interpreting the entire guest operating system, rather than running it on the native hardware, results in a significant performance drawback. When emulating a PC architecture on a PC, a compatibility layer that enables the target code to be run directly on the host processor, can reduce the performance penalty. On the other hand, virtualization partitions hardware in multiple contexts, thus enabling running multiple operating systems on the same hardware resources simultaneously. Several virtualization approaches exist[6]:

- Bare-metal Architecture
- Para-virtualization
- Full Virtualization, also known as Transparent Virtualization, or Hosted Architecture

The Bare-metal Architecture approach runs the guest operating system in Ring 0, the most privileged protection level in x86 architectures. Running multiple operating systems in the same protection level could potentially result in one of the systems compromising the other. Clearly, this approach is not acceptable for resource owners. The Para-virtualization approach needs to modify the host system by interposing a hypervisor between the operating system and the hardware. The hypervisor then takes on the Ring 0 and the operating system must be explicitly ported to run in Ring 1. These modifications to the host operating system exclude this approach. The Full Virtualization approach has performance drawbacks, but is the most secure and thus chosen. As shown in Figure 2, the virtual machine allows a guest operating system to run as an application in the host operating system. The virtual machine emulates the underlying hardware, thus creating a secure sandbox that allows an application written for one operating system, e.g. Linux, to be executed in another, e.g. Windows.

The performance penalties are mitigated by kernel support that enables it to run most of the target application code directly on the host processor, thus achieving near native speed. Regarding security, the virtual machine is a user space process that cannot do any harm to the host system as long as the permanent storage is protected properly. If the virtual machine is destroyed by a malicious application, the host system is not affected, and the virtual machine can start afresh. The following such solutions exist for the Windows platform:

- VirtualPC
- VMWare
- VirtualBox

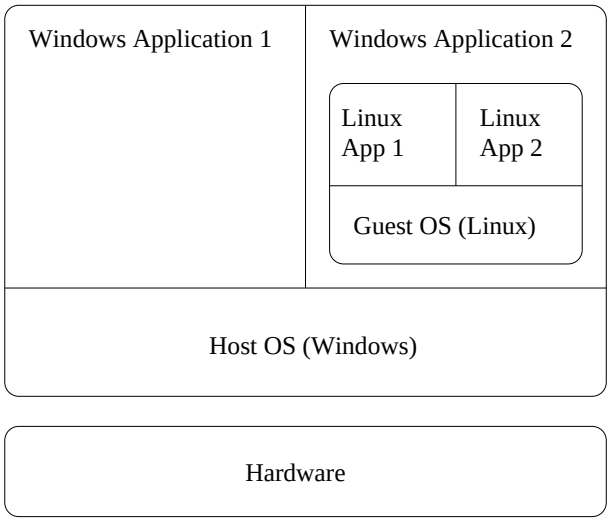


Figure 2: Full Virtualization

- Qemu + Qemu Accelerator Module

Having downloaded and installed one of the virtual machines, a resource owner only needs to install a screen saver that starts the virtual machine upon activation and a tailor-made Linux image that is capable of running the Grid resource software automatically. In this manner, when the resource goes into screen saver mode, the virtual machine is activated and the Linux guest operating system is booted. The details of how to Grid-enable the hosted Linux system are explained next.

3.2.1 The MiG Linux Guest OS

As explained above, all that is required for a resource to join MiG, is to create a grid user account and support for incoming SSH and outgoing HTTPS. So basically, the MiG Linux image can be built using any Linux distribution that runs an x86 system. Since the virtual machine provides a standardized virtualized set of hardware, compatibility amongst the wide range of different hardware setups on the resources will not be an issue. The main concerns with respect to the distribution is the size and the start-up time. Both issues matter only for practical reasons, the size should be minimized to avoid an excessively large download, and, naturally, the start-up time should be minimized as much as possible. In order to circumvent the missing inbound Internet access on resources that use NAT, it must be ensured that all communication is initiated by the resource. In MiG, this was easily integrated by small changes that only apply for sandboxes. In addition, it allows for directing jobs that users point out as Public Resource Computing jobs directly to a free sandbox. The generic MiG Linux Image consists of a kernel and a Ram-disk that altogether take up less than 3 MB. An online generator modifies the generic image by giving it a unique resource name and a session id needed for requesting a job. Further, the resource owner can choose the size of a hard disk image file to provide as storage for the sandbox. Thus, certified resource owners can have a complete image built with a unique key allowing the sandbox

to automatically request and execute Grid jobs. In the standard MiG model, the identity of a resource requesting a job is verified by keeping the public SSH key of the resource in the MiG system and copying all job files to the resource over SSH. The sandbox model however, is modified to use a pull model on the resource where all files are transferred using HTTPS. Hence, a firewall in front of a resource only needs to be open for HTTP and HTTPS to allow the resource to run Grid jobs.

3.2.2 Runtime Environments

Once the basic sandbox is in place, it is possible to execute user applications within the virtual Linux machine. Many applications can be passed as executables from the Grid job and these need no further components to execute. Other, commonly used, applications may benefit from a preinstalled runtime environment, such as they are found on ordinary Grid resources.

Installing runtime environments in the sandboxed environment could be done as on a conventional resource, which however would require the PC owners to personally maintain the sandboxed Linux distribution; this model is obviously not desirable. Alternatively the sandbox image could be distributed with the initial Linux image, but this would greatly increase the size of the distribution image and in addition be a very static model.

The chosen solution allows individual research groups to maintain runtime environments for the sandboxed resources and at the same time allows the individual PC owners to control which runtime environments are downloaded and at which time. The runtime environments are kept in individual virtual disk-partitions, in the form of a single file. The PC owner can download individual runtime environments from the VGrids, MiG's notion of a virtual organization, that maintain the runtime environments and when a job that uses a runtime environment is received by the sandboxed resource, the virtual Linux machine will mount the file system that contains the runtime environment. This way each runtime environment is kept isolated from the rest of the system, and can easily be built and maintained by the research groups that need them to be available for their executions.

3.3 Cycle-Scavenging using Java Applets

As explained above, all that is required for a PRC resource to join MiG is a sandbox and support for outgoing HTTPS. The previous solution presented above requires installation of non standard software to activate and execute the sandbox. In this model, "MiG One-Click"[1] the work imposed on the resource donor is taken to the extreme: No software install is needed.

To reach our stated goal of no Grid specific software installation and no modification of the donated machines firewall settings, we are forced to use software which is an integrated part of a common Internet connected resource.

We found that amongst the most common software packages for any PC type platform there is a Java enabled web browser. The web browser provides a common way of securely

[1]The URL accessed to activate the web browser as a sandboxed MiG Java resource is called "MiG One-Click", as it requires one click to activate it.

communicating with the Internet, which is allowed by almost all firewall configurations of the resources we target.[2]

The web browser provides us with a communication protocol, but it does not by itself provide a safe execution environment, however all of the most common graphics enabled web browsers have support for Java applets that are capable of executing Java byte-code located on a remote server.

The Java applet security model, ASM, prevents the Java byte-code executed in the applet from harming the host machine and thereby provides the desired sandbox effect for us to trust the execution of unknown binaries on donated resources.

The choice of web browsers and Java applets as the execution framework results in some restrictions on the type of jobs that may be executed in this environment:

- Applications must be written in Java
- Applications must apply to ASM
- The total memory usage is limited to 64 MB including the Grid framework
- Special methods must be used to catch output
- Special methods must be used for file access

By accepting the limitations described above, a web browser can be turned into a Grid resource simply by entering a specific URL. This triggers the load and execution of an applet which acts as our Grid gateway and enables retrieving and executing a Java byte-code based Grid job. The details of this process are described next.

3.3.1 The Applet Grid Resource

Several changes to the Grid middleware are needed to allow Java applets to act as Grid resources. First of all the Grid middleware must support resources which can only be accessed through a pull based model, which means that all communication is initiated by the resource, i.e. the applet. This is required because the ASM rules prevents the applet from initiating listening sockets, and to meet our requirement of functioning behind a firewall with no Grid specific port modifications. Secondly, the Grid middleware needs a scheduling model where resources are able to request specific type of jobs, e.g. a resource can specify that only jobs which are tagged to comply to the ASM can be executed.

The Java applet technology makes it is possible to turn a web browser into a MiG sandbox without installing any additional software. This is done automatically when the user accesses the MiG One-Click web page, which loads an applet into the web browser. This applet functions as a Grid resource script and is responsible for requesting pending jobs, retrieving and executing granted jobs, and delivering the results of the executed jobs to the MiG server.

To make the applet work as a resource script, several issues must be addressed. First of all ASM disallows local disk access. Because of this both executables and input/output files

[2] Resources located behind firewalls that do not support outgoing HTTPS is considered out of range for this PRC, however it is not unseen that outbound HTTPS is blocked.

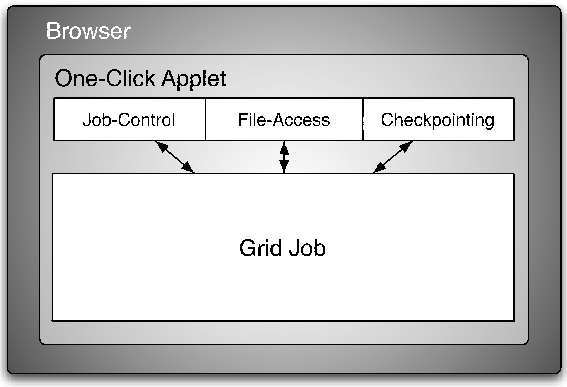


Figure 3: The structure of a One-Click job

must be accessed directly at the Grid storage. Secondly only executables that are located at the same server as the initial applet are permitted to be loaded dynamically. Thirdly text output of the applet is written to the web browser's Java console and not accessible by the Grid middleware.

When the applet is granted a job by the MiG server, it retrieves a specification of the job which specifies executables and input/output files. The applet then loads the executable from the Grid, this is made possible by the MiG server which sets up an URL from the same site as the resource applet was originally loaded which points to the location of the executables. This allows unknown executables to be loaded and comply with the ASM restrictions on loading executables. Figure 3 shows the structure of a One-Click job.

Executable jobs that are targeted for the MiG One-Click model must comply with a special MiG One-Click framework, which defines special methods for writing `stdout` and `stderr` of the application to the MiG system[3]. Normally the stdout and stderr of the executing job is piped to a file in the MiG system, but a Java applet, by default, writes the stdout and stderr to the web browsers Java console. We have not been able to intercept this native output path. Input and output files that are specified in the job description must be accessed directly at the Grid storage unit since the ASM rules prohibits local file access. To address this issue the MiG One-Click framework provides file access methods that transparently provide remote access to the needed files. Note that the MiG system requires input files and executables to be uploaded to the MiG server before job submission which ensures that the files are available at the Grid storage unit.

In addition to the browser applet a Java console version of the MiG resource has been developed, to enable the possibility of retrieving and executing MiG One-Click jobs as a background process. This requires only a Java virtual machine. To obtain the desired security model, a customized Java security policy is used, which provides the same restrictions as the ASM.

[3]The result of a MiG job is the stdout/stderr and the return code of the application that is executed.

3.3.2 Checkpointing

PRC resources will join and leave the Grid dynamically, which means that jobs with large running time have a high probability of being terminated before they finish their execution. To avoid wasting already spent CPU-cycles a checkpointing mechanism is build into the applet framework. Two types of checkpointing have been considered for inclusion, transparent checkpointing and semi-transparent checkpointing.

Transparent Checkpointing All to the authors known transparent checkpoint mechanisms provided to work with Java, require the JVM to be replacement or access to the /proc file system on Linux/Unix operating system variants, as the default JVM does not support storing program counter and stack frame. Since our goal is to use a web browser with the Java applet as a Grid resource neither of those solutions are satisfactory, since both the replacement of the JVM and access to the /proc file system violates the Java applet security model. Furthermore most PRC resource will be running the Windows operating system which do not support the /proc file system.

Semi-transparent Checkpointing Since transparent checkpointing is not applicable to the One-Click model, we went on to investigate what we call semi-transparent checkpointing. Semi-transparent checkpointing covers that the One-Click framework provides a checkpoint method for doing the actual checkpoint, but the application programmer is still responsible for calling the checkpoint method when the application is in a checkpoint safe state.

The checkpoint method stores the running Java object on the MiG server through HTTPS. Since it can only store the object state, and not stack information and program counters, the programmer is responsible for calling the checkpoint method at a point in the application, where the current state of the execution may be restored from the object state only. To restart a previously checkpointed job, the resource applet framework first discovers that a checkpoint exists and then loads the stored object.

3.4 MiG Features

To improve and simplify the sandboxes further, MiG contains two components that apply: Strong scheduling and remote file access.

3.4.1 Scheduling

Contrary to the majority of the existing Grid middlewares, where several levels of scheduling results in jobs being submitted to a resource where another level of scheduling takes place, MiG makes the scheduling for fairness much simpler as the local scheduling comes before the Grid scheduling. Thus, a single job is never left waiting a long time for CPU cycles once it has been submitted to a resource.

Existing Screen Saver Science systems all target problems that have many, usually millions, of independent tasks that often run for tens or hundreds of hours. Once a task has been assigned to a computer it will be processed while the computer is in screen saver mode. Processing is suspended if the screensaver is suspended and similarly resumed again along

with the screensaver. An artefact of this model is that one never knows when the result of a given task is ready, and it is very hard to determine if the task has been lost or if it is simply only allowed to proceed very slowly.

For applications such as computational chemistry this model is very poorly suited. The number of tasks is usually in the tens or hundreds and it is often the case that analyses of the results can only start once the result of every task is in.

This is easily addressed by putting an upper time limit on each job, and if the time limit is exceeded, the job is resubmitted to another resource. However, in order to schedule a job with a deadline to a screen saver resource, we need to know how long the resource is available, i.e. how much time it takes before the screen saver is deactivated. To predict the available time slot of a screen saver resource, we use exponential average on an hourly basis, which has proved to converge against the actual resource idle time quite fast.

3.4.2 Remote File Access

One difficulty that users report when using Grid is file access, since files that are used by Grid jobs must be explicitly uploaded to a Grid storage element and result files must also be downloaded explicitly. The MiG model introduces home catalogs for all Grid users, and all file references are relative to this home catalog. This eliminates all naming problems, since MiG provides one simple access entry to a user's home catalog. Furthermore, using the MiG Remote File Access library [1], applications running on a Linux resource - as the ones used in the MiG-SSS model - can, transparently and without recompiling or relinking applications, access application input and output files remotely, thus only downloading needed data and only uploading modified data.

The same ideas have been implemented in the One-Click Model which also provides transparent remote file access to the jobs that are executed. The MiG storage server supports partial reads and writes, through HTTPS, of any file that is associated with a job. When the resource applet accesses files that are associated with a job, a local buffer is used to store the parts of the file that are being accessed. If a file position which points outside the local buffer is accessed, the MiG server is contacted through HTTPS, and the buffer is written to the MiG server if the file is opened in write mode. The next block of data is then fetched from the server and stored into the buffer and finally the operation returns to the user application. The size of the buffer is dynamically adjusted to utilize the previously observed bandwidth optimally.

3.4.3 Block size estimation

To achieve the optimal bandwidth for remote file access it is necessary to find the optimal block size for transfers to and from the server. In this case the optimal block size is a trade off between latency and bandwidth. We want to transfer as large a block as possible without excessive latency penalty since the chance of transferring data that will not be used increases with the block size.

We define the optimal block size bs_{opt} as the largest block where a doubling of the block size does not double the time to transfer it. This can be expressed the following way:

$$t(x) * 2 > t(x * 2) \qquad \forall x < bs_{opt} \tag{1}$$

$$t(x) * 2 < t(x * 2) \qquad \forall x > bs_{opt} \tag{2}$$

$$t(x) \ = time\, to\, transfer\, block\, of\, size\, x$$

We do not want block sizes below bs_{opt} as the time t used to transfer a block of size x is less than doubled when the block size is doubled. On the other hand we don't want 'too large' block sizes as we do not know if the retrieved data is going to be used or discarded due to a seek operation beyond the end of the local buffer.

As the One-Click resources can be placed at any sort of connection, and the bandwidth of the connection thus may differ greatly from one resource to another, it is not possible to use a fixed block size and reach a good ratio between bandwidth and latency at an arbitrary type of connection.

The simplest approach would be to use a fixed bs_{opt} based on empirical tests on the most common connections.

A less trivial, but still simple, approach would be to measure the time it takes to connect to the server and then choose a block size which ensures the transfer time of that block to be a factor of x larger than the time to connect, to make sure that the connection overhead does not exceed the time of the actual data transfer.

The chosen approach is to estimate bs_{opt} from the time spent transferring block $x-1$ with the time of transferring block x, starting with an initial small[4] block size bs_0 and then doubling the block size until a predefined cutoff ratio CR is reached. After each data transfer the bandwidth bw_x is calculated and compared to the bandwidth of the previous transfer bw_{x-1}. If the ratio is larger than the predefined CR:

$$\frac{bw_x}{bw_{x-1}} > CR \tag{3}$$

then the block size is doubled:

$$bs_{x+1} = bs_x * 2 \tag{4}$$

As the block size is doubled in each step the theoretical CR to achieve bs_{opt} should be 2, since there is no incentive to increase block size once the latency grows linearly with the size of the data that is transferred.

However in reality, one need to get a CR below 2 to achieve bs_{opt}. This is due to the fact that all used block sizes are powers of 2, and one cannot rely on the optimal block size to match a power of 2.

Therefore to make sure to get a block size above bs_{opt} you need a lower CR. Empirical tests showed that a CR about 1.65 yields good results, see section 4.2

Additional extensions include adapting to the frequency of random seeks in the estimation of the CR. A large amount of random seeks to data placed outside the range of the current buffer will cause new blocks to be retrieved in each seek. Therefore the block size should be lowered in those cases to minimize the latency of each seek.

[4] An initial small block size gives a good result as many file accesses applies to small text files such as configuration files.

4 Experiments

To test the One-Click model we established a controlled test scenario. Eight identical Pentium 4, 2.4 GHz machines with 512 MB ram were used for tests.

4.1 One-Click as concept

The test application used is an exhaustive algorithm for folding proteins written in Java. This was changed to comply with the applet framework.

A protein sequence of length 26 was folded on one machine, which resulted in a total execution time of 2 hours, 45 minutes and 33 seconds. The search space of the protein was then divided into 50 different subspaces using standard divide and conqueror techniques. The 50 different search spaces were submitted as jobs to the Grid, which provides an average of 6 jobs per execution machine and 2 extra jobs to prevent balanced execution. The search spaces on their own also provide unbalanced execution as the valid protein configurations vary from one search space to another and thus results in unbalanced execution times. The experiment was made without checkpointing the application. The execution of the 50 jobs completed in 29 minutes and 8 seconds, a speedup of 5.7 for 8 machines. While this result would be considered bad in a cluster context it is quite useful in a Grid environment.

To test the total overhead of the model, a set of 1000 empty jobs was submitted to the Grid with only one One-Click execution resource connected. The 1000 jobs completed in 19935 seconds, which translates to an overhead of approximately 20 seconds per job.

4.2 File access

To achieve the best bandwidth cutoff ratio CR several experiments has been made. In the experiments a 16 MB file was read 100 times by the One-Click resource on a 20 Mb/s broadband Internet connection. All experiments start with an initial block size of 2048 (2^{11}) bytes. The first experiment was run with a CR of 0, which means that the block size is doubled in every transfer. The result is shown in figure 4.

The figure shows how the latency starts to raise dramatically between block size 2^{18} and 2^{20} and the bandwidth to latency ratio starts to fall at those block sizes. The bandwidth to latency ratio between block size 2^{18} and 2^{20} lies in the interval from 1.25 to 1.75. Based on these observations we performed the same test with a CR of 1.5. The result is shown in figure 5.

This shows that a CR of 1.5 is too low as block sizes of 2^{21} occur and we want the block sizes to be between 2^{18} and 2^{20} to limit the maximum latency. Therefore the CR must be between 1.5 and 1.75. The test was then run with CR 1.55, 1.60, 1.65, 1.70 and 1.75. The result is shown in figure 6.

We observe that a CR of 1.75 is too high, as only a few block sizes of 2^{19} occur and no block sizes of 2^{20} occurs. A CR of 1.55 results in a few block sizes of 2^{21} which is above the block sizes we want. 1.60 represents the block sizes we want and block size 2^{20} is well represented. A CR of 1.65 represents block size 2^{19} well and a few block sizes of 2^{20} is reached as well, and a CR 1.70 represents block size 2^{20} but no block sizes of 2^{21} are represented. We choose a CR of 1.65 as block size 2^{20} is considered the braking

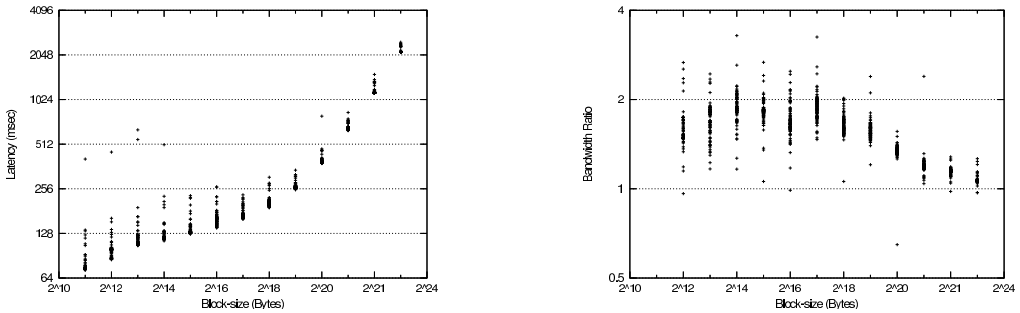


Figure 4: The left figure shows the latency as a function of the block size, the right figure shows the bandwidth ratio $\frac{bw_x}{bw_{x-1}}$ as a function of the block size. Between block size 2^{18} and 2^{20} the latency starts to raise and the bandwidth ratio starts to fall. This is where the cutoff is chosen to avoid excessive raise in latency

point where the latency starts to grow excessively, therefore we do not want it to be to well represented, but we want it to be represented, which is exactly the case at a CR of 1.65.

To verify the previous finding that the CR value should be 1.65 a test application, which traverses a 16 MB file of random 32 bit integers was developed. First the application was tested against the framework, where fixed block sizes were used, and then the application was tested against the framework, where the dynamic block sizes with a CR of 1.65 were used. The results are shown i figure 7.

The experiment shows, as expected, that the execution time decreases as the block sizes increase in the experiments with static block sizes. The execution time in the experiments with the dynamic block sizes all reside around 256 seconds[5] which are satisfactory, as this shows that compared to largest static buffer size of interest[6], the execution time loss using a dynamic buffer size is at most a factor of four. The reader should note that this type of application is the worst case for dynamic buffer sizing as all the data are read sequentially.

[5]With the exception of 3 runs, which are classified as outliers

[6]The largest static buffer of interest is 2^{17} as this is where the time gained by doubling the buffer, levels out.

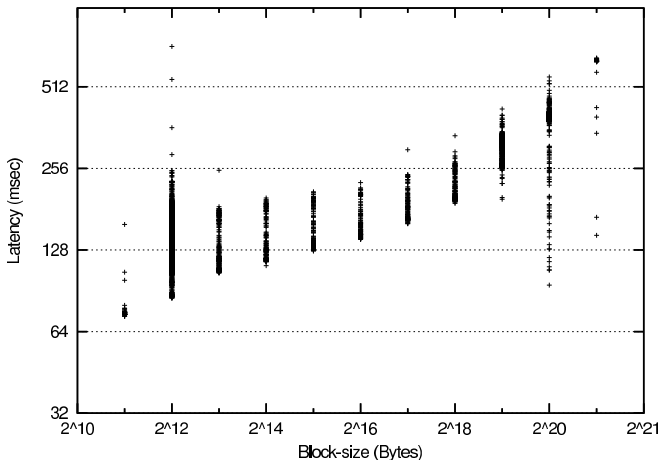


Figure 5: The latency as a function of the block size with CR 1.5

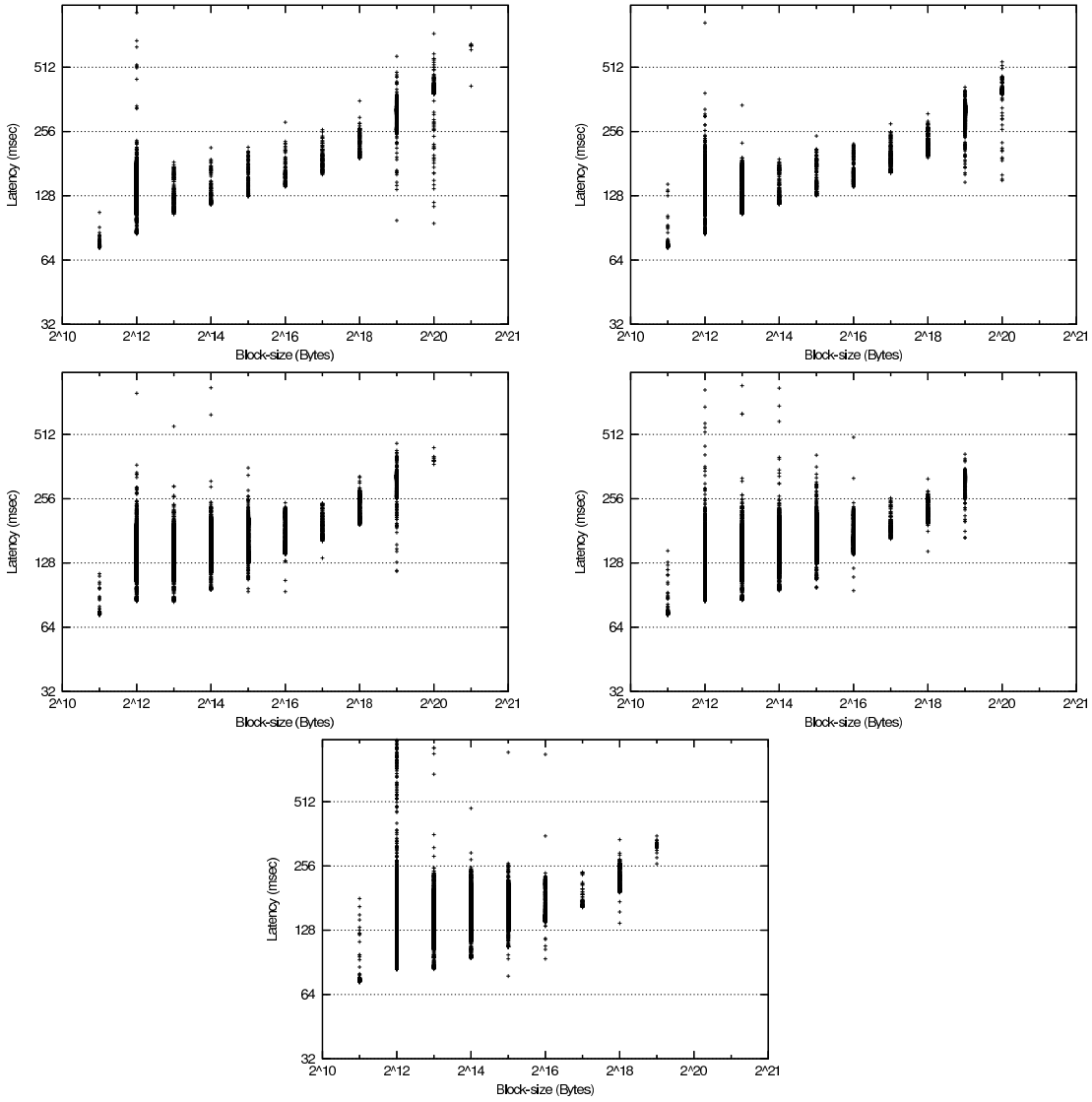


Figure 6: The latency as a function of the block size with CR 1.55, 1.60, 1.65, 1.70, 1.75

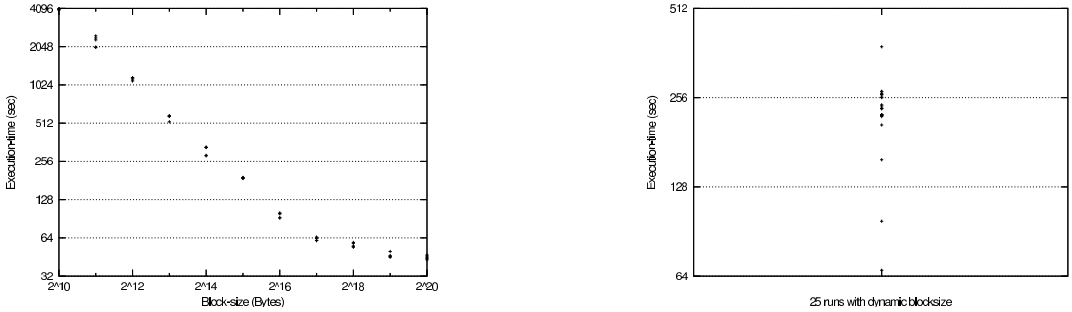


Figure 7: The execution time as a function of the block size and the execution time with dynamic block sizes and a CR of 1.65

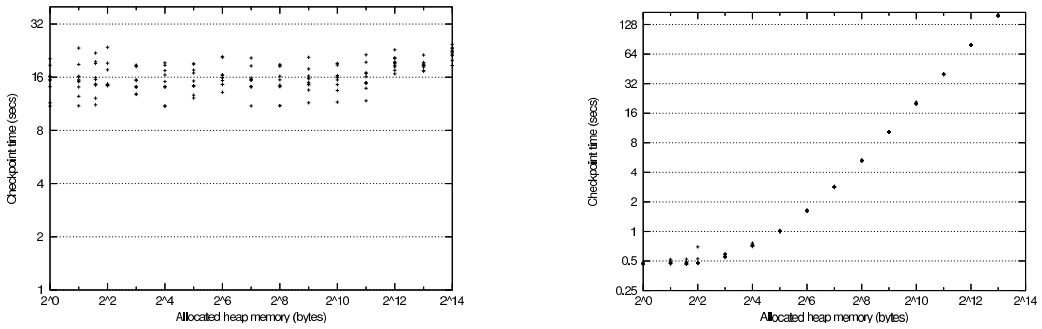


Figure 8: The time spent checkpointing on a 20 Mb/s and a 2048/412 kb/s Broadband Internet

If the integers were read in random order, the dynamic buffer size execution would perform much better.

4.3 Checkpointing

The next obvious performance issue is to test the overhead of performing a checkpoint operation within a process. This was tested by submitting jobs that allocate heap memory in the range from 0 kB to 8192 kB. Each job first allocates X kB, where X is in the order power of 2, and does 10 checkpoints, which saves the entire heap space. The performance was first tested on a 20 Mb/s broadband Internet connection.

The test was then repeated using a more modest 2048/512 kb/s broadband Internet connection. The result of these tests is shown in figure 8.

In the first test, using the 20 Mb/s connection, the checkpoint time is constant as the memory size grows. We can conclude from this, that the overhead of serializing the Java object is dominating compared to the actual network transfer time. The opposite is the case when we examine the results of the 2048/512 kb/s connection. Here we see that the time spent grows linearly with the size of the allocated memory, from which we may conclude that on a 512 kb/s connection the bandwidth is, not surprisingly, the limiting factor.

The experiments also show that the dynamic block sizes approach increases the execution time by of factor of four compared to the execution time reached with the largest static block size in a worst case scenario.

The building checkpointing mechanism has an overhead of 15 seconds per checkpoint on a 2.4GHz P4 and the One-Click framework overall is causing approximately 20 seconds of overhead to each execution, compared to local execution. Despite of this, a considerable speedup is reached in the presented protein experiment.

5 Conclusion

This work has shown how to eliminate the factors that have previously impeded the fusion of Public Resource Computing and Grid Computing to effectively utilize idle CPU cycles from desktop machines for any kind of Grid job.

The prohibiting factors include NAT-hidden resources, means to utilize Windows desktops, the workload required by a non-expert resource owner to install and manage all resource software, and the security issues involved with installing a large software base on the resource.

Using sandboxing technology we have successfully eliminated all of these limitations. Two models exist, the MiG-SSS and the One-Click model. In the MiG-SSS model, resource donors download a bundle consisting of a screen saver, a virtual machine, and a special MiG Linux image in order to share their idle resources. The MiG One-Click model lowers the workload imposed on the resource owner to only requiring a visit on a web page.

The MiG system has proved flexible enough to easily deal with computers behind network address translators, and mobile processes and automatic resubmission of jobs solve the problem with resources that are cut off the network or leave the screen saver mode.

Using the MiG-SSS approach, a desktop computer volunteers as a Grid resource upon screen saver activation, and as soon as the screen saver is deactivated, the executing job either stops or migrates. Thus, the resource owner is completely unaffected by the Grid job. The users submitting jobs to the public resources gain access to a virtualized Linux environment where standard Linux applications can run unmodified.

In the One-Click framework, resource owners can contribute without installing any client software at all. By using Java Applet technology, the resource owner simply points a Java-enabled browser to the MiG One-Click URL which will load an applet acting as a Grid resouce. Once the user of the donated computer wishes to stop the execution, the browser is simply closed down or pointed to another URL, and the execution stops. The MiG system eventually detects this event, by a timeout, and resubmits the job to another resource, where the job is resumed from the latest checkpoint that was made.

The use of Java applets provides a secure sandboxed executing environment that prevents the executing Grid jobs from harming the donated machine. The disadvantage of this approach is that all jobs must be written in Java and in addition comply with the presented framework, including the Java Applet Security Model. However the modifications that are needed to port an existing Java application are limited to using special methods for stdout and stderr, applying to the Java applet security model, and using the One-Click framework for remote file access. The One-Click framework also includes means to provide semi-transparent checkpointing of the applications at runtime.

Experiments have been performed to find the optimal block size for the remote file transfer that the framework includes. The experiments show that doubling the block size in each transfer gives the optimal tradeoff between bandwidth and latency as long as the CR is below 1.65.

References

[1] Rasmus Andersen and Brian Vinter, *Transparent remote file access in the minimum intrusion grid*, WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (Washington, DC, USA), IEEE Computer Society, 2005, pp. 311–318.

[2] David P. Anderson, *Boinc: A system for public-resource computing and storage*, GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (Washington, DC, USA), IEEE Computer Society, 2004, pp. 4–10.

[3] Allan Bricker, Michael Litzkow, and Miron Livny, *Condor Technical Summary, Version 4.1b*, 1992.

[4] Brad Calder, Andrew A. Chien, Ju Wang, and Don Yang, *The entropia virtual machine for desktop grids*, VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments (New York, NY, USA), ACM, 2005, pp. 186–196.

[5] Renato J. Figueiredo, Peter A. Dinda, and José A. B. Fortes, *A case for grid computing on virtual machines*, ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems (Washington, DC, USA), IEEE Computer Society, 2003.

[6] Ian Foster and Carl Kesselman, *The grid: blueprint for a new computing infrastructure*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, November 1998.

[7] Henrik Hoey Karlsen and Brian Vinter, *Minimum intrusion grid - the simple model*, WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (Washington, DC, USA), IEEE Computer Society, 2005, pp. 305–310.

[8] Brian Vinter, *The Architecture of the Minimum intrusion Grid (MiG)*, Communicating Process Architectures 2005, sep 2005, pp. –.

Appendix G

Publication 7

Recent Developments in Grid Technology and Applications. G.A. Gravvanis, J.P. Morrison, H.R. Arabnia and D.A. Power, Editors **Brian Vinter, Rasmus Andersen, Martin Rehr, Jonas Bardino, and Henrik Karlsen: Towards a Robust and Reliable Grid Middleware**

In: ISBN:
Editor:

Chapter

TOWARDS A ROBUST AND RELIABLE GRID MIDDLEWARE

Brian Vinter, Rasmus Andersen, Martin Rehr, Jonas Bardino and Henrik Karlsen
University of Copenhagen, Department of Computer Science

Abstract

This chapter describes the philosophy behind a new Grid model, the Minimum intrusion Grid, MiG. The idea behind MiG is to introduce a 'fat' Grid infrastructure which allows much 'slimmer' Grid installations on both the user and resource side. Components that differentiate MiG from Globus and similar models include a zero-size Grid code base that is required to be installed on client or resource computers, mandatory payment and pricing of Grid services, end-to-end anonymity between consumers and producers and approximating the perception of a PC by means of Grid computing. In addition MiG provides a new, simpler approach to Virtual Organizations and a seamless integration of the Public Resource Computing model into Grid.

1. The idea behind the Grid

In the mid 1990's when Grid first was introduced, the idea of getting computing resources from a socket in the wall helped create a giant hype around Grid-computing. Since then one could argue that, as money has been poured into Grid research, the ambition level of the same research has dropped proportionally; to the point where Grid today often appears to be merely web-services in another wrapping! Grid research and Grid systems seem to be severely limited by initial choices made towards location, naming and security aspects in Grid. Most, or even all, of these were not obviously wrong, or were not at all wrong at the time they were made, but in the end we have ended up with a Grid model that is a long way from providing computing resources from a socket in the wall, particularly for commercial enterprises and in particular for private users.

Grid computing is just around the top of the hype-curve, and while large demonstrations of Grid middleware exist, including Globus toolkit[8] and NorduGrid ARC[9], the tendency in Grid middleware these days is towards a less powerful model, Grid services, than what was

available previously. This reduction in sophistication is driven by a desire to provide more stable and manageable Grid systems. While striving for stability and manageability is obviously right, doing so at the cost of features and flexibility is not so obviously correct.

The Minimum intrusion Grid, MiG, is a project that aims to design a new platform for Grid computing which is driven by a stand-alone approach to Grid, rather than integration with existing systems. The goal of the MiG project is to provide a Grid infrastructure where the requirements on users and resources alike, to join Grid, are as small as possible – thus the minimum intrusion part. While striving for minimum intrusion, MiG still seeks to provide a feature rich and dependable Grid solution.

2. Grid Middleware

The driving idea behind the Minimum intrusion Grid project is to develop a Grid middleware that allows users and resources to install and maintain a minimum amount of software to join the Grid. MiG will seek to allow very dynamic scheduling and scale to a vast number of processors. As such MiG will close the gap between the existing Grid systems and popular "Screen Saver Science" systems, like SETI@Home.

2.1 Philosophy behind MiG

"The Minimum intrusion Grid", this really is the philosophy - we want to develop a Grid middleware that makes as few requirements as possible. The working idea is to ensure that a user needs only a signed x509 certificate, trusted by Grid, and a web-browser capable of secure HTTP, HTTPS. A resource on the other hand must also hold a trusted x509 certificate and in addition create a user – the Grid user – who can use secure shell, ssh, to enter the resource and once logged on can open HTTPS connections to the outside. The requirements then become:

	User	**Resource**
Must have certificate	Yes	Yes
Must have outbound HTTPS	Yes	Yes
Must have inbound SSH	No	Yes[1]

Table 1. Requirements for using MiG

3. What's wrong with the classic Grid systems?

While there are many Grid middleware systems available most of them are based on, or descendents of, the Globus toolkit. Thus the description below addresses what the author believe to be shortcomings in the Globus toolkit, and not all issues may be relevant to all Grid systems.

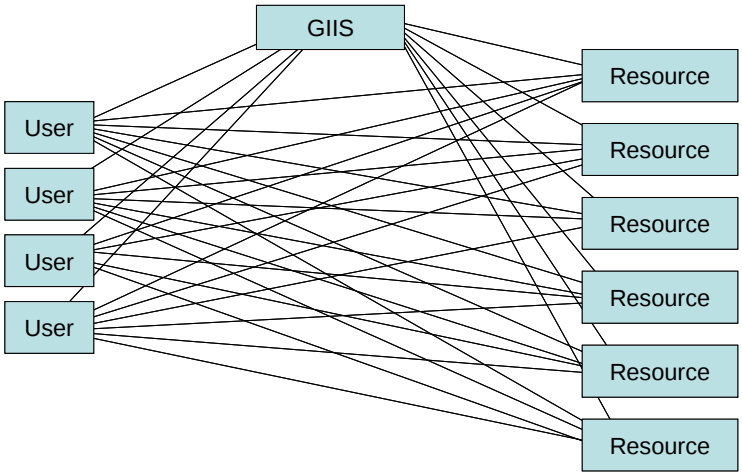
[1] Some resource models in MiG, known as sandboxes, do not even require inbound ssh.

3.1 Single point of failure

Contrary to popular claim, all existing Grid middlewares hold a central component that, if it fails, requires the user to manually choose an alternative. While the single point of failure may not truly be a single point, but comply with some level of redundancy, none of the components scale with the size of the Grid. Thus even for a Grid of infinite size a finite number of failures are required to make the overall Grid fail

3.2 Lack of scheduling

The classic Grid systems perform a job-to-resource mapping. However, an actual scheduling with a metric of success is not available. Work is underway in this in the community scheduler[16] but for this scheduler to work, the resources need to be exclusively signed over to Grid, i.e. a machine can not be accessed both through Grid and a local submission system.



Scheduling layout of the Globus Grid model

3.3 Poor scalability

The time taken to perform the job-to-resource mapping in the current systems scales linearly with the number of sites that are connected. This is already proving to be a problem in NorduGrid, which is one of the largest known Grids, though only 36 sites are connected. Imagining tens of thousands of connected sites is not likely. In the Grid service model scalability issues are more or less eliminated by absence of a single system view from a user perspective and thus forces the user to visit every site on the Grid before making a decision on where to place a new job.

3.4 No means of implementing privacy

The job submission API at the users machine communicates directly with all the potential sites, thus all sites know the full identity of all jobs on the Grid. This means that the grid user does not only reviles his or her full identity and intentions to the resource that end up executing the job, which in itself is bad enough, full disclosure is given to all participating resources in the Grid. Thus if one wish to make a map of ‘who-does-what’ on the Grid all that is needed is to place a resource on the Grid and log all incoming requests. There is not even a need to accept any jobs at all, simply to receive requests and the reject them.

3.5 No means of utilizing ‘cycle-scavenging’

Cycle-scavenging, or Screen Saver Science, utilizes spare CPU cycles when a machine is otherwise idle. This requires an estimate on how long the machine will be available and all classic Grid systems just assume that a free resource will be available indefinitely. Cycle scavenging in Grid has been partly demonstrated in NorduGrid by connecting a network of workstations running Condor, to NorduGrid, but Grid itself has no means of screen-saver science.

3.6 Requires a very large installation on each resource and on the user site

The middleware that must be installed on a resource to run NorduGrid, which is probably the most robust of the well known Grid middlewares, is more than 367 MB including hundreds of components. All of which must be maintained locally. This means that donating resources to Grid is associated with significant costs for maintenance; this naturally limits the willingness to donate resources.

3.7 Firewall dependency

To use the existing middlewares special communication ports to the resource must be opened in any firewall that protects a resource. This is an obvious limitation for growing Grid since many system administrators are reluctant towards such port-openings. One project that seeks to address this problem is the centralized-gateway-machine project under the Nordic Data Grid Facility[17] that receives jobs and submits them to the actual resource using SSH.

3.8 Highly bloated middleware

The existing middleware solutions provide a very large set of functions that are placed on each site, making the software very large and increasing the number of bugs, thus the need for maintenance, significantly.

3.9 Complex implementation using multiple languages and packages

Many current Grid middlewares have reused a large amount of existing solutions, for data-transfer, authentication, authorization, queuing, etc. These existing solutions are written in various languages and thus the Grid middleware uses more than 6 programming languages and several shell types, in effect raising the cost of maintaining the package further. The many languages and shells also limit portability to other platforms. The Globus project have acknowledged this as a problem and have switched entirely to Java for implementing their middleware.

4 Design Criteria for the Minimum intrusion Grid

All MiG component must design and implement a functional Grid system with a minimal interface between the Grid, the users, and the resources. The successful MiG middleware implementation holds the following properties.

4.1 Non-intrusive

Resources and users must be able to join Grid with a minimum of effort and with a minimum software installation. The set of requirements that must be met to join Grid must also be minimal. “Minimal” in this context should be interpreted rigidly, meaning that if any component or functionality in MiG can be removed from the resource or user end, this must be done, even if adding the component at the resource or user end would be easier.

4.2 Scalable

MiG must be able to contain tens of thousands, even millions, of resources and users without the size of the system impacts performance. Even individual PCs should be able to join as resources. For a distributed system, such as MiG, to be truly scalable it is necessary that the performance of the system is not reduced as the number of associated computers grows.

4.3 Autonomous

MiG should be able to perform an update of the Grid without changing the software on the user or resource end. Thus compatibility problems that arise from using different software versions should be eliminated by design. To obtain this feature it is necessary to derive a simple and well defined protocol for interaction with the Grid middleware. Communication within the Grid can be arbitrarily complex though since an autonomous Grid architecture allows the Grid middleware to be upgraded without collaboration from users and resources.

4.4 Anonymous

Users and resources should not see the identity of each other if anonymity is desired. This is a highly desirable feature for industrial users that are concerned with revealing their intentions to competing companies. A highly speculative, example could be two pharmaceutical companies A and B. Company A may have spare resources on a computational cluster for genome comparisons, while B may be lacking such resources. In a non-anonymous Grid model, company B will be reluctant to use the resources at company A since A may be able to derive the ideas of B from the comparisons they are making. However, in a Grid that supports anonymous users, A will not know which company is running which comparisons which makes the information far less valuable. In fact many comparisons will be likely to be part of research projects that map genomes and will thus reveal nothing but information that is already publicly available.

4.5 Fault tolerance

Failing machines or processes within the Grid should not stop users or resources from using the Grid. While obvious, the lack of fault tolerance is apparent in most Grid middlewares today. The consequences of lacking fault tolerance range from fatal to annoying. Crashes are fatal when a crashed component effectively stops users from running on Grid, i.e. a hierarchy of Meta Directory Servers.

If a resource that runs users' processes crash it becomes costly for the users that are waiting for the results of the now lost jobs. Finally crashes are merely annoying when a crashed component simply does not reply and thus slows down the users interactions with the Grid because of timeouts.

4.6 Firewall compliant

MiG must be able to run on machines behind firewalls, without requiring new ports to be opened in the firewall. While this requirement is quite simple to both motivate and state, actually coping within the restraints of this point may prove highly difficult.

4.7 Strong scheduling

MiG provides real scheduling, not merely job-placement, but it needs to do so without requiring exclusive ownership of the connected resources. Multi-node scheduling should be possible as should user-defined scheduling for dynamic subtasking. In effect MiG also supports meta-computing[2].

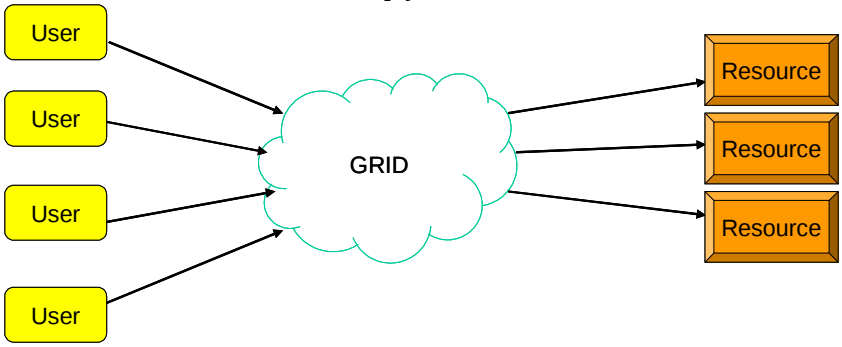
[2] Metacomputing is a concept that precedes Grid computing. The purpose of metacomputing is to create a large virtual computer for executing a single application.

4.8 Cooperative support

In order to improve the meta-computing qualities, MiG provides access to shared user-defined data-structures. Through these data-structures a MiG based Grid system can support collaborating applications and thus improve the usability of Grid.

5 The abstract MiG model

The principal idea behind MiG is to provide a Grid system with an overall architecture that mimics a classic, and proven, model – the Client-Server approach. In the Client-Server approach the user sends his or her job to the Grid and receives the result. The resources, on the other hand, send a request and receive a job. After completing the job the resource sends the result to the Grid which can forward the reply to the user.



The abstract MiG model

The Grid system should be disjoint from both the users and the resources, thus the Grid appears as a centralized black-box to both users and resources.

This model allow us to remain in full control of the Grid, thus upgrades and trouble shooting can be performed locally within Grid, rather than relying on collaboration from a large number of system administrators. In addition, moving all the functionality into a physical Grid system, lowers the entry level that is required for both users and resources to join, thus increasing the chances that more users and resources do join the Grid.

In MiG, storage is also an integrated component and users will have their own ‘home directory’ on MiG, which can be easily accessed and referenced directly in job-descriptions so that all issues with storage-elements and replica catalogues is entirely eliminated.

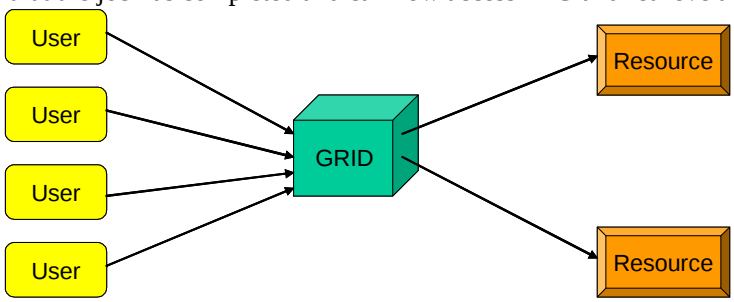
For a user to join, all that is required is an x509 certificate which is signed by a certificate authority that is trusted by MiG. Accessing files, submitting jobs and retrieving results can the all be done through a web-browser that supports certificate based HTTPS. As a result the user need not install any software to access Grid and if the certificate is carried on a personal storage device, e.g. a USB key, a user can access Grid from any internet enabled machine.

The requirements for resources to join MiG should also be an x509 certificate, but in addition the resource must create a Grid account in which Grid jobs are run. Initially MiG requires that this user can SSH into the account, some resources are run in a sandboxed pull mode instead but these are not described in this chapter..

6 The simple MiG model

In a simple version of the MiG model there's only a single node acting as the Grid. Clients and resources then communicate indirectly through that Grid-node. The interface between the user and Grid should be as simple as possible. The exact protocol remains a topic for investigation but, if possible, it will be desirable to use only the HTTP protocol or a similar widely used, and trusted, protocol. Towards the resources the protocol should be equally simple, but in this case, as we also desire that no dedicated Grid service is running on the resource, one obvious possibility is to use the widely supported SSH protocol.

When submitting a job, the user sends it to the Grid machine which stores the job in a queue. At some point a resource requests a job and the scheduler chooses a job to match the resources that are offered. Once the job is completed the results are sent back to MiG. The user is informed that the job has completed and can now access MiG and retrieve the results.



The simple MiG model

6.1 Considering the simple model

The simple model of course, is quite error-prone as the single Grid machine becomes both a single point of failure and a bottleneck which is not acceptable. The obvious solution is to add more Grid machines which can act as backup for each other.

7 The full MiG model

The obvious flaw in using the client-server model is that achieving robustness is inherently hard in a centralized server system where potential faults include:

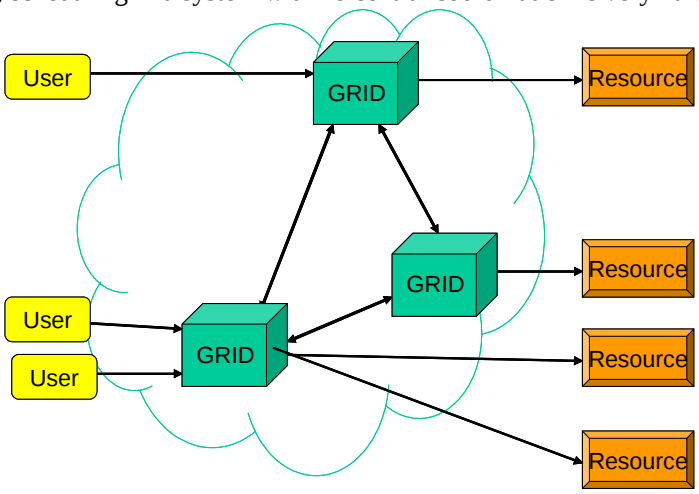
- Crashed processes
- Crashed computers
- Segmented networks
- Scalability issues

To correctly function in the presence of errors, including the above, error redundancy is needed. The desired level of redundancy is a subject to further investigations, but should probably be made dynamic to map the requirements of different systems. To address the performance issues Grid itself must be distributed so that users can contact a local Grid server. Thus workload will be distributed through the physical distribution of users.

Once a job arrives at a Grid server the server must ensure that the job is “deposited” at a number of other servers, according to the current replication rate. The user should not receive an acknowledgement of submission before the job has been correctly received and stored at the required number of servers.

Once a resource has completed a job the resource is expected to deliver the result. If, however, the client has not provided a location for placing the result, the resource can still insist on uploading the results. To facilitate this, the Grid should also host storage to hold results and user input-files, if a resource cannot be allocated at the time the client submits his job.

To facilitate payment for resources and storage a banking system should be implemented. To allow inter-organization resource exchange, the banking system should support multiple banks. Dynamic price-negotiation for the execution of a job is a very attractive component that is currently a research topic. Supporting price-negotiations in a system such as MiG where no central knowledge is available is an unsolved problem that must be addressed in the project. Likewise, scheduling in a system with no central coordination is very hard.



The full MiG model

7.1 Considering the full model

One topic for further investigations is: how do we schedule on multiple Grid servers? In principle we would prefer complete fairness, so that the order in which jobs are executed is not dependent on where they are submitted, i.e. to which MiG node. Such a full coordination between all nodes in MiG for each job-submission is not realistic since it will limit scalability, thus a model that allows scalability while introducing some level of load-balancing and fairness will have to be invented.

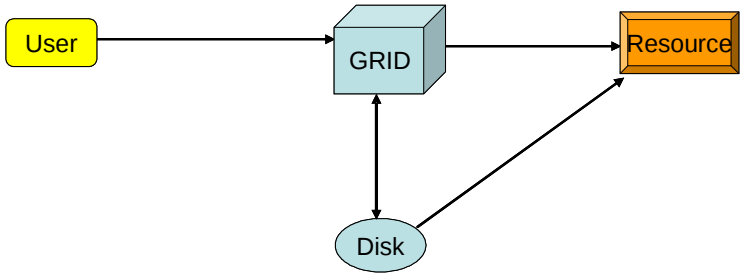
8 MiG Components

8.1 Storage in MiG

One difficulty that users report when using Grid is file access. Since files that are used by Grid jobs must be explicitly uploaded to a Grid storage element, result files must be downloaded equally explicitly. On the other hand it is a well known fact that the expenses associated with a professional backup strategy often prohibit smaller companies from implementing such programs, and relies on individual users to do the backup - a strategy that naturally results in a large loss of valuables annually. Some interesting statistics include:

- 80% of all data is held on PCs (Source, IDC)
- 70% of companies go out of business after a major data loss (Source, DTI)
- 32% of data loss is due to user error (Source, Gartner Group)
- 10% of laptops are stolen annually (Source, Gartner Group)
- 15% of laptops suffer hardware failure annually (Source, Gartner Group)

By using the Grid, we do not just gain access to a series of computational resources, but also to a large amount of storage. Exploitation of this storage is already known about from peer-to-peer systems, but under “well-ordered" conditions it can be used for true Hierarchal Storage Management, HSM. When working with HSM the individual PC or notebook only has a working copy of the data which is then synchronized with a real dataset located on Grid. By introducing a Grid based HSM system, MiG offers solutions to two important issues at one time; firstly Grid jobs can now refer directly to the dataset in the home-catalog thus eliminating the need for explicit up- and down-loads of files between the PC and Grid. Second, and for many smaller companies much more importantly, we can offer a professionally driven storage-system with professional backup solutions, either conventional backup systems or, more likely, simple replica based backup - the latter is more likely because disks are becoming rapidly less expensive and keeping all data in three copies is easily cheaper than a conventional backup-system and the man-power to run it. A Grid based HSM system also allows small companies to outsource the service while medium and large companies can chose to either outsource or implement a Grid HSM in-house. Thus by introducing Grid based HSM, Grid can offer real value to companies that are not limited by computational power and these companies will thus be "Grid integrated" when Grid becomes the de-facto IT infrastructure.



MiG Storage support

8.2 Scheduling

Scheduling in Grid is currently done at submission-time and usually a scheduled task is submitted to a system where another level of scheduling takes place. In effect the scheduling of a job provides neither fairness for users nor optimal utilization of the resources that are connected to the Grid, and the current scheduling should probably just be called job-placement. Furthermore, the current model has a built in race-condition since the scheduling inquires all resources and submits to the one with the lowest time-to-execute. If two or more jobs are submitted at the same time they will submit to the same resource, but only one will get the expected timeslot. The MiG model makes scheduling for fairness much simpler as the local scheduling comes before the Grid scheduling in the proposed model.

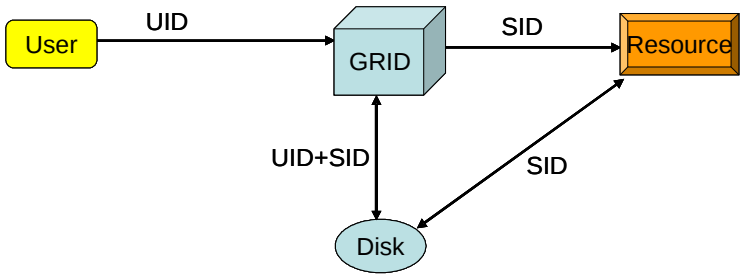
Scheduling for the best possible resource utilization is much harder and of much more value. The problem becomes one that may be described as: given the arrival of an available resource, and an existing set of waiting jobs, which job is chosen for the newly arrived resource so that the global utilization will be as high as possible?

The above is in the common case where jobs are more frequent than resources, in the rare case that resources are more abundant than jobs, the same problem is valid on the arrival of a job.

When scheduling a job, future arrivals of resources are generally not known, i.e., we are dealing with an on-line scheduling problem. On-line scheduling is an active research area, initiated as early as 1966 and continued in hundreds of papers, see and for a survey. This problem, however, differs from all these on-line scheduling problems investigated previously in that the resources, not the jobs, arrive over time in the common case. The problem also has some similarity with on-line variable-sized bin packing , but again with a twist that has not been considered before; the bins, not the items to be packed, arrive on-line.

8.3 Security and Secrecy

In Grid, security is inherently important, and the MiG system is designed be at least as secure as the alternative systems. The simple protocols and minimal software based on the resources make this goal easy to achieve, but still the mechanisms for security must be investigated. Secrecy is much harder and is currently not addressed in Grid. Privacy will mean much towards achieving secrecy but other issues are also interesting topics of research. I.e. if a data file is considered valuable, e.g. a genomic data sequence, how can we hold the contents of that file secret to the owner of the resource? In other words, can MiG provide means of accessing encrypted files without asking the users to add decryption support to his application?

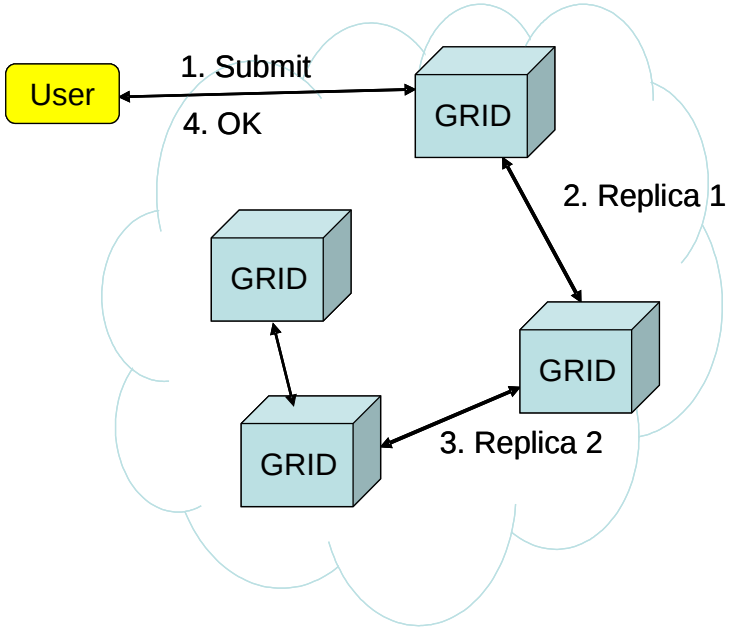


Anonymity and security model

8.4 Fault-tolerance

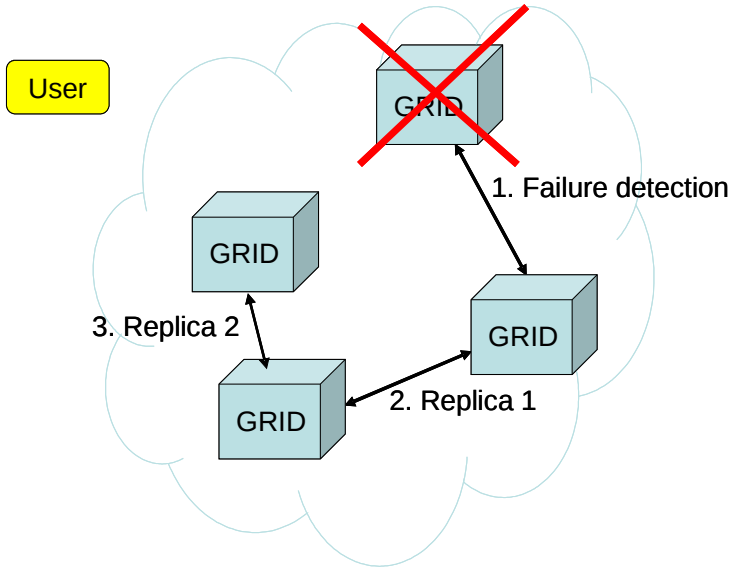
In a full Grid system errors can occur at many levels and failures must be tolerated on MiG nodes, resources, network connections and user jobs. Any single instance of these errors must be transparent to the user. More complex errors of course, or combinations of the simple errors, cannot fully be hidden from the users, i.e. if a user is on a network that is segmented from the remaining internet we can do nothing to address this.

Achieving fault tolerance in a system such as MiG is merely a question of never loosing information when a failure occurs, e.g. keeping redundant replicas of all information. shows how a submitted job is replicated when it is submitted.



Replicating a new job

Recovering from a failure is then a simple matter of detecting the failure and restoring the required number of replica's as shown in where the number of replicas is three.



Recovering from a failure

8.5 Load balancing and economics

Load balancing in distributed systems is an interesting and well investigated issue. However load balancing for, potentially, millions of resources while maintaining a well defined measure of fairness is still an unsolved issue. However adding economics to the equation actually makes this easier. Since MiG should support a market oriented economy, where the price for executing a job is based on demand and supply, this introduces a simple notion of fairness which is that resources should optimize their income while users should minimize their expenses.

In case there are more jobs than resources, which is the common case, the next job to execute is the job that is willing to pay most for the available resource. In case two or more jobs bid the same for the resource the oldest of the bidders is chosen.

In the rare case that there are more resources offering their services than there are jobs asking for a resource, the next available job is sent to the resource that will sell its resources cheapest. In case more resources bid at the same price, the one that have been waiting the longest wins the bid.

8.6 Shared data-structures for MiG

When people with little knowledge of Grid computing are first introduced to Grid, they often mistake it for meta-computing and expect the Grid to behave as one large parallel processor and not a large network of resources. This misunderstanding is quite natural, since such a Grid computing model would be highly desirable for some applications, of course most parallel applications cannot make use of such an unbalanced virtual parallel processor.

However, to support the applications that can make use of Grid as a meta-computing system, to address this MiG provides support for shared data-structures which are hosted on Grid.

Users who wish to utilize MiG for meta-computing may currently use one of four approaches

1. Shared files
2. MiGSpace communication
3. SQL database
4. CSP

The use of shared files are often seen in applications that are executed on a LAN and is quite simple, though not very powerful. Because, MiG works with a Grid based home-directory and runtime access to the files in this catalog, rather than a pure copy-semantics as commonly seen with Grid.

MiGSpace communication is in the tradition of Linda. Similar to the Linda tuple space, tuples provide the granularity for shared entities in MiGSpace. Single variable based granularity is ineffective in high latency environments. Communication latency is of significance to the task grain size in distributed shared memory systems. When communication latency increases, the grain size must be coarser to achieve good performance. Document based granularity like that found in xSpace is overly specialized. In contrast to JavaSpaces, tuples in MigSpace are flexible ordered collections of typed elements, similar to those found in Linda. Arrays, sets and matrices remain relatively common data structures in scientific computing, even though object oriented designs and languages have become more popular. MiGSpace supports matrices and arrays with its simple tuple approach. If desired is very easy to develop a tuple-to-object bridge. The following is a formal definition of tuples in MigSpace:

A tuple consists of finite collections of ordered typed elements. Each element can be an actual or a formal. The following notation for a tuple is used:

$<P_1, P_2, P_3, ...P_j>$,

where Pi is an element.

A tuple consists of only actuals. Actuals have a type and a value. The

following is an example of a tuple with three elements in which all are actuals:

$<1_{int}, "John"_{string}, 2_{int}>$

Applications may this write actuals to the MiGSpace and read by using formals. Since parallel programming with tuple-spaces is a well established paradigm, a large set of algorithms that are designed for use with tuple-spaces. MiGSpace extends the classic tuple-space model by introducing the option of reading and writing entire sub-spaces, this is introduced to help hide latency over wide area networks.

Spaces in MiGSpaces are implemented as files and thus inherit their full security model from the file-level security.

The SQL interface is quite straight forward, applications may access a MiG hosted SQL database through a MiG enabled ODBC interface that seamlessly wraps all SQL queries in the MiG security mechanisms and forwards the request to the MiG server-side SQL engine. Likewise the reply is wrapped in the security layer and returned to the ODBC layer at the execution host where the ODBC library translates the reply into standard ODBC format.

CSP, Communication Sequential Processes, is a well established model for designing and implementing concurrent applications[ref]. In CSP the communication mechanism is synchronous channels and in MiG-CSP these channels have been extended to a Grid

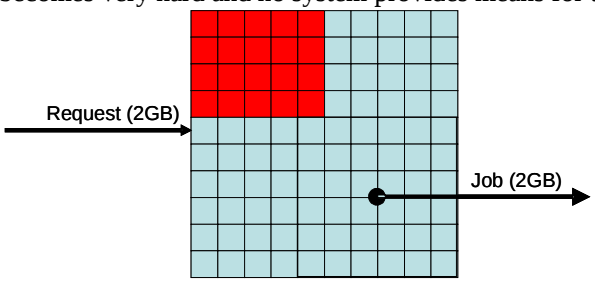
environment. The channels too are implemented as files and thus inherit the entire security model from the MiG file modes.

8.7 Accounting/Price-negotiations

Grid becomes really interesting once users can purchase resources on Grid, thus transforming Grid from a resource sharing tool into a market place. To support this vision, MiG does not only do accounting but also support a job bourse, where the price for a task can be dynamically negotiated between a job and a set of resources. Such dynamic price-setting is also a known subject, but combining it with load-balancing and fairness in a truly distributed system has not been investigated.

8.8 User defined scheduling

An advanced extension of the online-scheduling problem is the subtasking problem, where a job may be divided into many subjobs. If the subtasks have a natural granularity the task is trivial and known solutions exist, including functioning systems, such as SETI@Home. If, on the other hand, a subtask can be selected that solves the largest possible problem on the given resource, the problem becomes very hard and no system provides means for this today.



Dynamic sub-scheduling

When comparing with on-line bin packing, this variant of the problem has one further twist to it; the size of an item (a subtask) may depend on which other items are packed in the same bin, since the data needed by different subtasks may overlap.

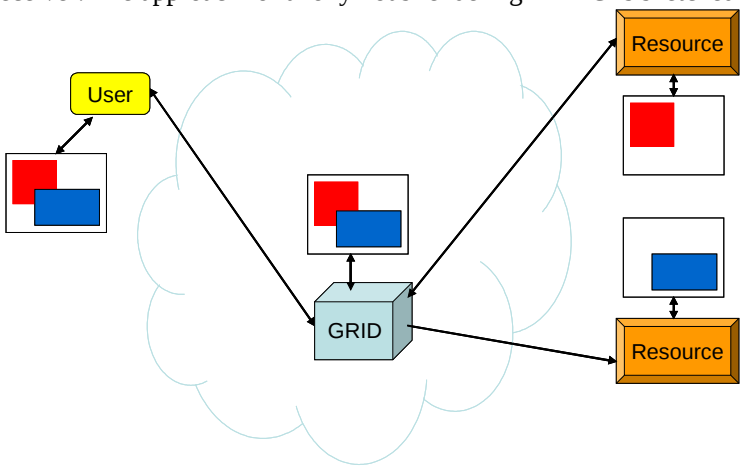
MiG has developed a model where a job can be accompanied with a function for efficient sub-tasking. The demonstration application for this will be a new version of the Grid BLAST application, which is used in Bio-Science for genome comparisons. The efficiency of BLAST depends on two parameters; input-bandwidth and available memory. We currently developing a dynamic subtasking algorithm that creates subjobs fitted for resources as they become available.

8.9 Graphics rendering on Grid

Currently Grid is used exclusively for batch job processing. However for Grid to truly meet the original goal of “computing from a plug in the wall”, graphics and interactivity is needed.

In this respect MiG makes things more complex than the existing middlewares since MiG insists on maintaining anonymity, e.g. we insist that a process can render output to a screen-buffer that it cannot know the address of.

The solution to this problem is similar to the storage model. A ‘per-user’ frame-buffer is hosted in the MiG infrastructure, and resources can render to this, anonymous, region. Users on the other hand can choose to import this buffer into their own frame-buffer and thus observe the output from their processes without the hosts of these processes knowing the identity of the receiver. The approach for anonymous rendering in MiG is sketched in .



Anonymous graphics rendering in MiG

9 Virtual Organisations

Facilitating the organization and work of Virtual Organizations is amongst the premiere advantages of Grid computing. In [4] it is stated that “the real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.” Resource sharing in this context is not only about files, but access to all kinds of resources like computational power, external data, software and specialized hardware. When sharing resources in loosely organized collaborations, such as virtual organizations, one need the ability to apply a number of rules and conditions which define the policy of the individual collaborations. Thus, seen at the utmost abstraction level, a virtual organization is a set of well defined individuals who share a well defined set of resources. While essential to Grid computing, Virtual Organizations are defined purely as concept in Grid computing and not strictly defined by a protocol or similar. The most widespread implementation of Virtual Organizations is the Virtual Organization Membership Service, VOMS, which essentially works by allowing the user to request a proxy-certificate from a given Virtual Organization, using that proxy-certificate the user may then continue to submit the desired job to a resource that accepts the VO proxy-certificate.

9.1 VOMS

VOMS work by allowing the user to request a proxy certificate that verifies VO membership from any VOMS server. If the VOMS server has information to the end that the requesting users is in fact member of the desired VO it returns the proxy-certificate. Using that proxy-certificate the user may then continue to submit the desired job to a resource that accepts the VO proxy-certificate. The process is shown in figure 2.

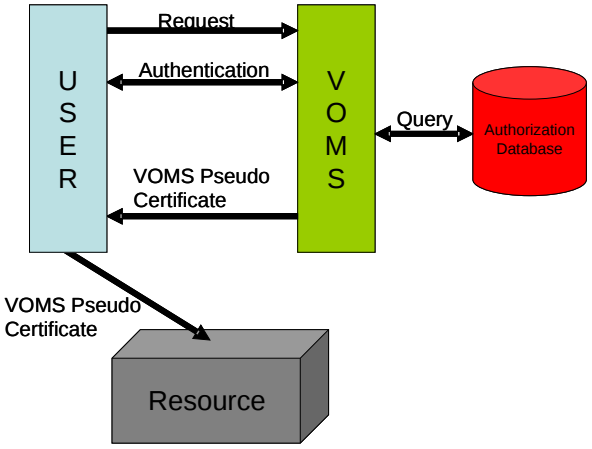


Figure 2. VO membership certificate in VOMS

When presented with a proxy-certificate the resource verifies that the user is member of an accepted VO and continues the authorization process. The resource may maintain a list locally with banned users and deny a user access even though VO membership has been confirmed by the VOMS server through the proxy-certificate.

9.2 CAS

Community Authorization Service, CAS, which is part of the Globus toolkit[8][7], seeks to provide a more fine grained access control than simply membership of a VO. CAS works by introducing a new abstraction level in the system called roles. Roles are similar to sub-groups in a VO except that roles are not only associated with a set of resources but also with the operations that may be performed on the resources, i.e. a specific role may only provide read privileges to a data-set but not write privileges. The process that implements this mechanism is identical to the overall VOMS process as shown in figure 2.

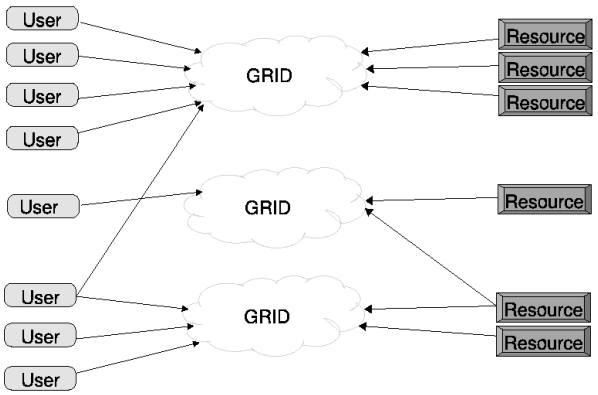
9.3. GridShib

GridShib[5] is a project which seeks to replace ordinary VOMS and CAS systems with a Shibboleth[1] based authorization model. One of the driving motivations for Grid-Shib is the same as one of the primary motivations for MiG, namely the need for user privacy.

9.4 VGrids

The MiG design has made it easy to obtain a lot of the Grid features that was previously very hard to implement. The development of MiG in general has greatly benefited from the knowledge and mistakes learned by the first middleware that appeared. The VOMS approach of proxy certificates is cumbersome and represents some concerns on manageability and security, as an example it is not possible to revoke a proxy-certificate. Solutions to these issues was discussed for some time in the MiG team. We seek a model that supports the anonymity required by MiG, and which does not introduce proxy-certificates, a concept which is entirely eliminated from the MiG design and which should not be re-introduced in order to support VOs. At the same time we also need to keep the anonymity between user and resource and provide the strong-scheduling capabilities found in MiG. Another important observation from real-world Grids is that many large resources are hard to connect to Grid since they run on user group quota allocations and often use a fair-share scheduling mechanism. In order to support access to a resource from Grid by two independent user groups, complex submission handling or even multiple Grid entry-points must often be introduced, both of which increase the complexity of managing resources towards Grid and thus decreases motivation to join a Grid system. We believe it is imperative to support the natural regulation mechanisms in local sites, and we also believe that local administration of Grid related options should be kept at a minimum, according to the project name, minimum intrusion grid. The proposed solution is an entirely different approach to virtual organizations. In its nature Grid seeks to allow a set of users to share a set of resources, while VOs seek to control which users of a Grid share which resources, in essence a VO becomes a subset of a Grid. With this in mind we choose Grids as our basic mechanism and treats a VO as Grid-whithin-a-Grid, or Virtual Grids, VGrids (figure 3). A VGrid appears to a user almost as an ordinary Grid, it has users, compute-resources and data-resources. As MiG seeks to hide much of the Grid complexity to the user, all data-resources a user has access to are presented in the form of a unified file-system, and VGrid data-resources appear as subdirectories in the users home-directory. Resources never see the identity of the users due to the anonymity feature of MiG. It therefore makes no sense for resources to maintain a local list of banned users as in traditional middleware. If a resource joins a VGrid it grants access to all users within it.

In fact, today there are no resources in the ordinary MiG Grid any longer, all resources are located in one or more VGrids.



VGrids are integrated by design.

The set of allowed users in a VGrid consist of two types, owners and members. All valid MiG users are allowed to create new VGrids and can then include any other user they know of as either co-owners of the VGrid or ordinary members of the VGrid. The user who creates it automatically becomes an owner of the new VGrid. Owners can add and remove other owners and members, but a VGrid must always have at least a single owner. Besides having the authorization to manage other owners and members an owner also has the privileges as regular members, that is to access the resources in the VGrid and the files belonging to the VGrid.

The authorization structure is hierarchical and as such similar to the structure of VO's. If you have owner or member rights of a VGrid you automatically have the same rights on all sub VGrids. This means that an owner of VGrid V0 is also an owner of V0/V1 and V0/V1/V2 but not the other way round, i.e. some VGrids may have owners that are not even members of the parent VGrid if this is desired the MiG servers and the client never communicate directly with the resources, everything goes through the central MiG servers. This means that the input files needed to execute a job must first be uploaded to the server from where the resource can retrieve the file before executing the job. After a job has been executed the outputfiles are uploaded by the resource to the server and the user can download the file or use it in new job submissions or simply leave the file at the server where it may be safely stored.

To support file sharing between members of a VGrid, a directory is created on the server where all members are allowed to read and write. It looks just like any other directory in the members home directories, but the files within it are readable and writable by all VGrid members.

One of the observed complications with the VOMS model is the problem of having two user groups with each their allocation on a large resource that both accesses their allocation through Grid. In the VGrid model this becomes extremely easy, and both allocation quotas and fair share scheduling is supported by default and without requiring any administration on the local resource. Upon creating a VGrid the owners of that VGrid can add resources to it. As in the original MiG model a resource is simply an ordinary user account that allow incoming ssh and outgoing https. The administrator of a resource allocation simply creates an account on the resource, as he would do with a new member of his research team, and registers that account with the VGrid. From that point the VGrid can use the resource, but locally at the resource the VGrid use appears simply as the use of an ordinary, untrusted, user.

VGrids have a number of additional advantages and features compared to the VOMS approach. One is custom optimized job submission When a VGrid is created two web-page references are automatically generated within the MiG namespace. A public page that can be accessed by all Internet users where the owners can promote and publish information about the project, and a private page which is only accessible by members of the VGrid. Within the private page a custom job submission page may be created, using HTML. This page can be optimized to the special purpose of the VGrid, often in the form of an application portal to Grid execution by the VGrid owner using a set of predefined names for the HTML controls in a HTML form which has the MiG server as target. The MiG server generates a job description file based on the values of the HTML controls when it receives the form and submits it to the execution queue of the VGrid on behalf of the user. All usual HTML controls

are thereby available for the VGrid owners when they create their specialized submit page as well as Cascading Style Sheets (CSS), a technology often used in conjunction with HTML to create a set of web pages with a consistent look.

Another special component is the VGrid Monitor and Statistics. A Grid monitor with a snapshot of the current state of the Grid and a web page with various statistics e.g. the total number of jobs the Grid has processed are two features that have been a part of most Grid middlewares for some time. Besides an overall monitor and statistics page for the complete Grid, the same information is available for the separate VGrids. The default is that the VGrid monitor and statistics pages are only accessible by members of the VGrid, but the owners can change this to make the information public available.

VGrids also allow for easy information sharing. A WIKI is a special kind of website where users can easily add and edit pages and content. It is a very simply way for a group of people to collaborate and together create and maintain information. Perhaps the most known WIKI is the wikipedia [3] where Internet users together have created an encyclopedia that at the time of this writing has more than 950.000 articles and more than 900.000 registered users that maintain the information. This is big-scale collaboration! The fundamental technology behind a WIKI is HTTP and HTML, technologies as MiG is build upon and compatible with. It is possible to install a WIKI back-end on the MiG server and let the owners of MiG VGrids make WIKI functionality available on VGrid pages that can be accessible to the public or to members only.

11 Conclusions

The purpose of this paper is to motivate the work on a new Grid middleware, the Minimum intrusion Grid, MiG. MiG is motivated in a set of claimed weaknesses of the existing Grid middleware distributions, and a desire to develop a model for Grid computing that is truly minimum intrusion.

The proposed model will provide all the features known in today's Grid systems, and a few more, while lowering the requirements for a user to simply having an X.509 certificate, and for a resource to have a certificate and create a Grid-user who can access the resource through SSH.

While MiG is still in its very initial stage, users can already submit jobs and retrieve their results, while maintaining complete anonymity from the resource that executes the job.

References

[1] R. L. Graham, Bounds for Certain Multiprocessing Anomalies, Bell Systems Technical Journal, vol 45, 1563--1581, 1966

[2] Y. Azar, On-Line Load Balancing, Online Algorithms: The State of the Art, Springer-Verlag, 1998, A. Fiat and G. J. Woeginger (ed.), Lecture Notes in Computer Science, vol. 1442

[3] J. Sgall, On-Line Scheduling, Online Algorithms: The State of the Art, Springer-Verlag, 1998, A. Fiat and G. J. Woeginger (ed.), Lecture Notes in Computer Science, vol. 1442

[4] J. Csirik, An On-Line Algorithm for Variable-Sized Bin Packing, Acta Informatica, 26, pp 697--709, 1989.

[5] J. Csirik and G. Woeginger, On-Line Packing and Covering Problems, Online Algorithms:

The State of the Art, Springer-Verlag, 1998, A. Fiat and G. J. Woeginger (ed.), Lecture Notes in Computer Science, vol. 1442

[6] L. Epstein and L. M. Favrholdt, On-Line Maximizing the Number of Items Packed in Variable-Sized Bins, Eighth Annual International Computing and Combinatorics Conference (to appear), 2002

[7] I. Foster. The Grid: A New Infrastructure for 21st Century Science. Physics Today, 55(2):42-47, 2002.

[8] I. Foster, C. Kesselman. The Globus Project: A Status Report. Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop, pp. 4-18, 1998.

[9] P. Eerola et al. "Building a Production Grid in Scandinavia". IEEE Internet Computing, 2003, vol.7, issue 4, pp.27-35.

[10] R. Fielding et al, RFC2616 Hypertext Transfer Protocol -- HTTP/1.1, http://www.rfc.net/rfc2616.html, The Internet Society, 1999 .

[11] T. Ylonen, SSH - Secure login connections over the internet, Proceedings of the 6th Security Symposium, p 37, 1996.

[12] S. F. Altschul et al., Basic local alignment search tool, J. Mol. Biol. 215:403-10, 1990.

[13] G. Barish and K. Obraczka, World Wide Web Caching: Trends and Techniques, IEEE Communications Magazine Internet Technology Series, May 2000.

[14] Minimum intrusion Grid - The Simple Model, Henrik H Karlsen and Brian Vinter, in proc. of ETNGRID 2005 (to appear)

[15] Transparent Remote File Access in the Minimum Intrusion Grid, Rasmus Andersen and Brian Vinter, in proc. of ETNGRID 2005 (to appear)

[16] The Community Scheduler Framework, http://csf.metascheduler.org, 2005.

[17] The Nordic Data Grid Facility, NDGF, www.ndgf.org, 2003.

[18] Data Clinic, http://www.dataclinic.co.uk/data-backup.htm